Ministry of Education and Science of Ukraine
Ternopil Ivan Puluj National Technical University

Faculty of Computer Information System and Software Engineering
(full name of faculty)

Computer Science
(full name of department)

QUALIFYING PAPER

For the degree of

bachelor
(degree name)

topic: **Development of a Mobile Application in the Form of a 2D-game for Training of User's Multitouch Technique**

Submitted by: fourth year student _____ , group ICH-42

specialty 122 Computer Science

(code and name of specialty)

	(signature)	Mokhliss Khalid (surname and initials)
Supervisor	(signature)	R. Hromiak (surname and initials)
Standards verified by	(signature)	H. Shymchuk (surname and initials)
Head of Department	(signature)	I.Bodnarchuk (surname and initials)
Reviewer	(signature)	(surname and initials)

Ternopil
2021

Ministry of Education and Science of Ukraine
Ternopil Ivan Puluj National Technical University

Faculty Faculty of Computer Information System and Software Engineering
(full name of faculty)

Department Computer Science
(full name of department)

APPROVED BY
Head of Department
I. Bodnarchuk
(signature) (surname and initials)
«25» Jan 2021

ASSIGNMENT
for QUALIFYING PAPER

for the degree of bachelor
(degree name)

specialty 122 Computer Science
(code and name of the specialty)

student Mokhliss Khalid
(surname, name, patronymic)

1. Paper topic Development of a Mobile Application in the Form of a 2D-game for Training of User's Multitouch Technique

Paper supervisor Roman Hromiak
(surname, name, patronymic, scientific degree, academic rank)

Approved by university order as of «*01*» *Feb* 2021 № 4/7-62
2. Student's paper submission deadline June, 25th
3. Initial data for the paper

4. Paper contents (list of issues to be developed)

INTRODUCTION; 1.1 The game idea; 1.2 The game UML; 1.3 Character design; 2 TEHNOLOGIES USED; 2.1 Unity; 2.2 Adobe animate; 2.3 C-sharp; 3 THE BUILDING PROCESS; 3.1 Animation process; 3.2 Unity scenes building; 3.3 The scripts; 3.4 Animation process; 4 LIFE SAFETY, FUNDAMENTALS OF LABOR PROTECTION; 4.1 The electromagnetic fields effects; 4.2 First-aid care provision for a broken bone; CONCLUSION ; REFERENCES

5. List of graphic material (with exact number of required drawings, slides)

6. Advisors of paper chapters

Chapter	Advisor’s surname, initials and position	Signature, date	
		assignment was given by	assignment was received by
Life safety, fundamentals of labor protection			

7. Date of receiving the assignment 25 Jan 2021

TIME SCHEDULE

LN	Paper stages	Paper stages deadlines	Notes

Student	(signature)	Mokhliss Khalid (surname and initials)
Paper supervisor	(signature)	Roman Hromiak (surname and initials)

ABSTRACT

Development of a Mobile Application in the Form of a 2D-game for Training of User's Multitouch Technique // Qualification work of the educational level "Bachelor" // Mokhliss Khalid // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information System and Software Engineering, Department of Computer Science // Ternopil, 2021 // P. – 46, Fig. – 30, References – 11.

As we know the human been is so interesting, he can really do more than one activities at the same time, but he also make some mistakes doing so. And we call it also the human multitasking.

A game which help you to train your brain for the multitasking by giving you instead of one character you must to handle the movement of two characters to avoid a bunch of obstacles.

Doing so even the kids with concentration disabilities can enjoy the game also training the mind to reach a normal concentration amount. Which is at least 10 min continue.

LIST OF SYMBOLS

OOP – Object-oriented programming

ADHD – Attention Deficit Hyperactivity Disorder

2D – two dimension

3D – three dimension

CLI – common language infrastructure

ISO – International Organization for Standardization

CONTENT

INTRODUCTION 6
1.1 The game idea 7
1.2 The game UML 7
1.3 Character design 13
2 TEHNOLOGIES USED 14
2.1 Unity 14
2.2 Adobe animate 16
2.3 C-sharp 18
3 THE BUILDING PROCESS 20
3.1 Animation process 20
3.2 Unity scenes building 23
3.3 The scripts 26
3.4 Animation process 39
4 LIFE SAFETY, FUNDAMENTALS OF LABOR PROTECTION 41
4.1 The electromagnetic fields effects 41
4.2 First-aid care provision for a broken bone 43
CONCLUSION 45
REFERENCES 46

INTRODUCTION

The ADHD Attention Deficit Hyperactivity Disorder is characterized by developmentally inappropriate levels of inattention, hyperactivity, and impulsive behavior. The condition can often cause significant impairment in daily functioning, both at home and at school. The Brain Balance Program focus is on understanding the struggles children experience and helping them develop and strengthen the connections needed to help those struggles. Brain Balance does not require that a child has a medical diagnosis, nor do we clinically diagnose medical conditions. The unique symptoms your child is coping with can improve through a unique combination of physical, sensory, and cognitive activities that help build stronger connections across different regions of the brain. Research has shown that these symptoms may be linked to weak connections across different regions of the brain. The good news is, we know that the brain can change in a way that can lead to improvements in these symptoms.

The Brain Balance program exercises and activities are uniquely designed to help strengthen and build new connections by using a combination of physical, sensory, and cognitive activities. These new, efficient, and effective connections in the brain lead to improvements in our ability to complete cognitive tasks and in our executive functioning. We use what we learn in the Personal Assessment to set up your child’s program. By doing this for each child, individually, we create a customized program that addresses any deficits in functions or skills your child may have.

The frequency and duration of these tasks, activities, and exercises are tailored based on your child’s needs. So I was thinking about how I could help those peoples with this disability then I get the idea to create a game that can increase the mind concentration.

1 THE GAME DESIGN

1.1 The game idea

At this game we start with two characters at the bottom of a well, while the water is filling up the place the characters must start climbing to the top by bouncing around and dashing the obstacles, the game is endless mode so there is no end only and just only if you lose by crashing one of the characters with an obstacle or touched by water.

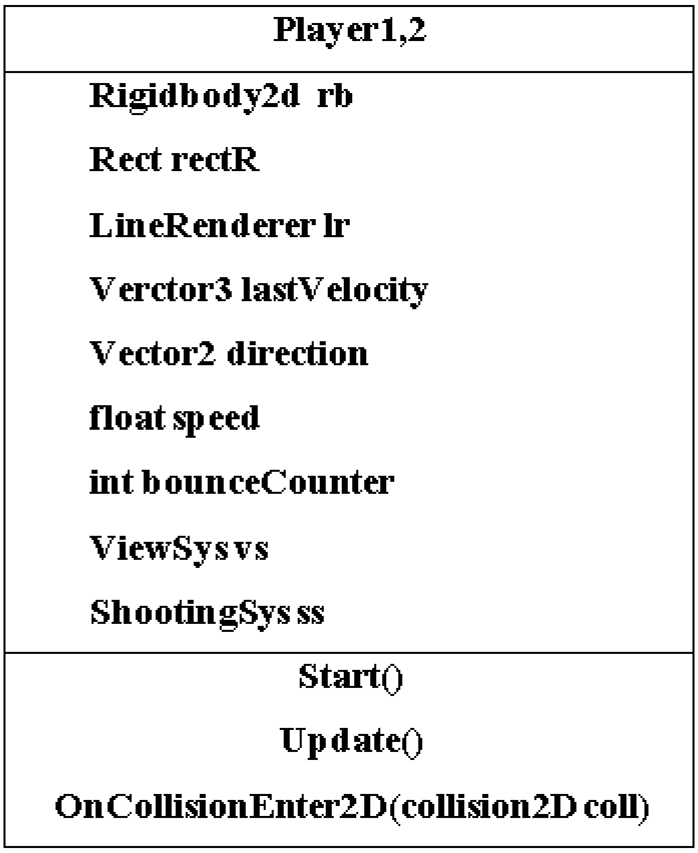
One of the characters can support the fire obstacle but not the bubble obstacle, rather than the other character can support the opposite obstacle, but also both characters die if they touch the bottom of the well (water).

By this way the player is going to be obligated to focus with all the obstacles also

the player must control two characters instead of one which make the game controls more hard, helping the player to increase the focus for those who have a concentration disabilities, also training the human multitasking for those who are normal.

1.2 The game UML

Following figures show the class diagrams for developed game.



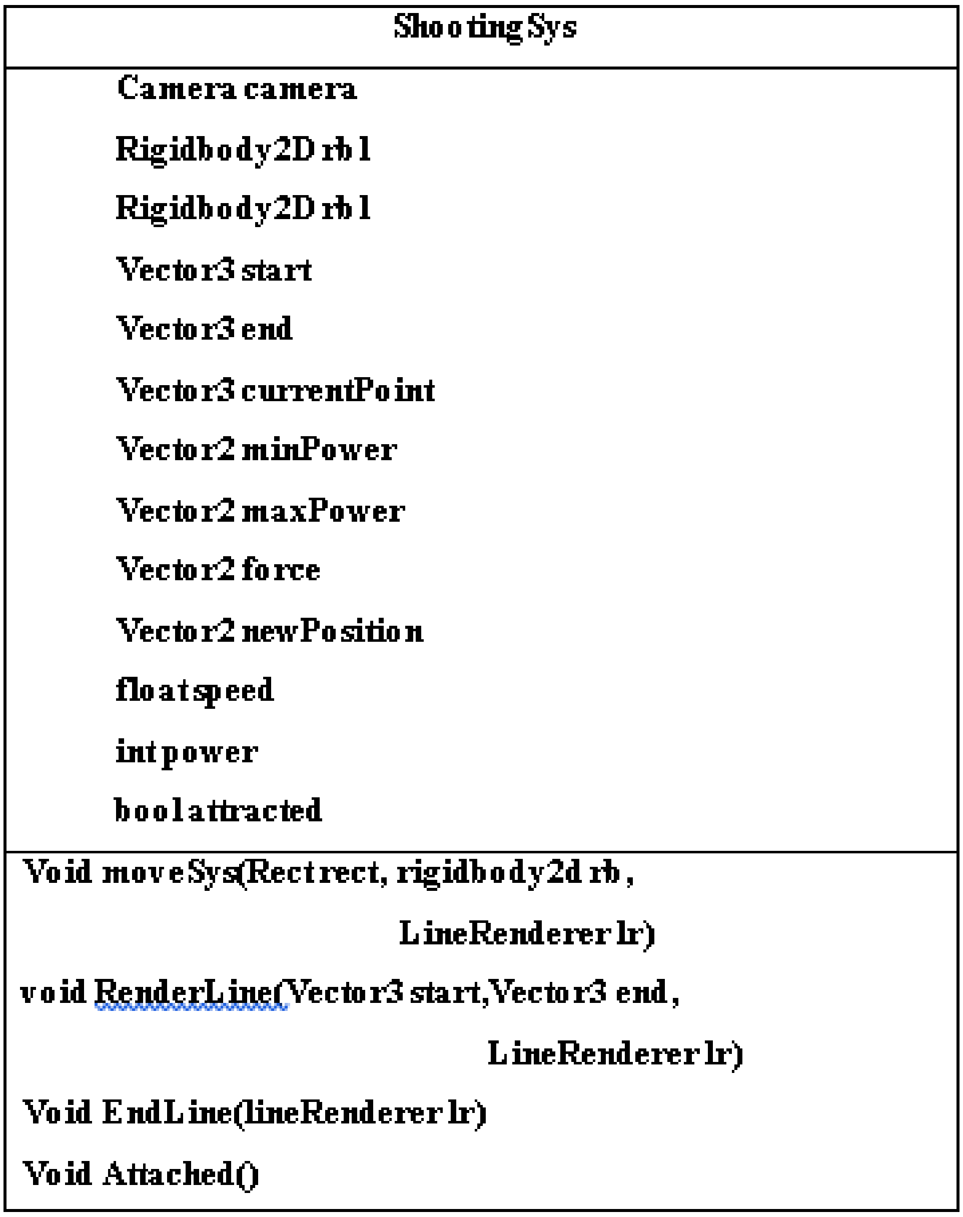
Player1,2

- Rigidbody2d rb
- Rect rectR
- LineRenderer lr
- Verctor3 lastVelocity
- Vector2 direction
- float speed
- int bounceCounter
- ViewSys vs
- ShootingSys ss

- Start()
- Update()
- OnCollisionEnter2D(collision2D coll)

Figure 1.1 - Player 1, 2 class

This is two class player 1 and player 2, I made for them one table because it is the same classes just with the rigidbody2D affected difference I tried to encapsulate those classes by using one interface then create those classes while irritating from the interface but unity don’t support multiple inheritance, which C# can use.



ShootingSys

Camera camera

Rigidbody2D rb1

Rigidbody2D rb1

Vector3 start

Vector3 end

Vector3 currentPoint

Vector2 minPower

Vector2 maxPower

Vector2 force

Vector2 newPosition

float speed

int power

bool attracted

Void moveSys(Rect rect, rigidbody2d rb, LineRenderer lr)

void RenderLine(Vector3 start,Vector3 end, LineRenderer lr)

Void EndLine(lineRenderer lr)

Void Attached()

Figure 1.2 - ShootingSys class

This class is responsible on the characters movement mechanics so the bouncing calculation and the direction all done in this class each time the player drag and drop his finger on the screen to lunch the characters.

Also, it is helping to provide the automatic movement when the first character want to cross the top screen and the second character always stuck t the bottom of the screen.

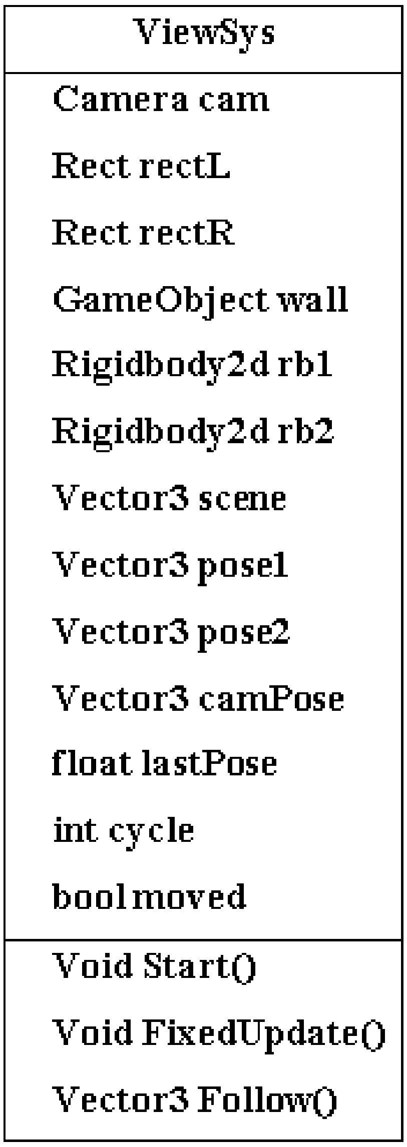
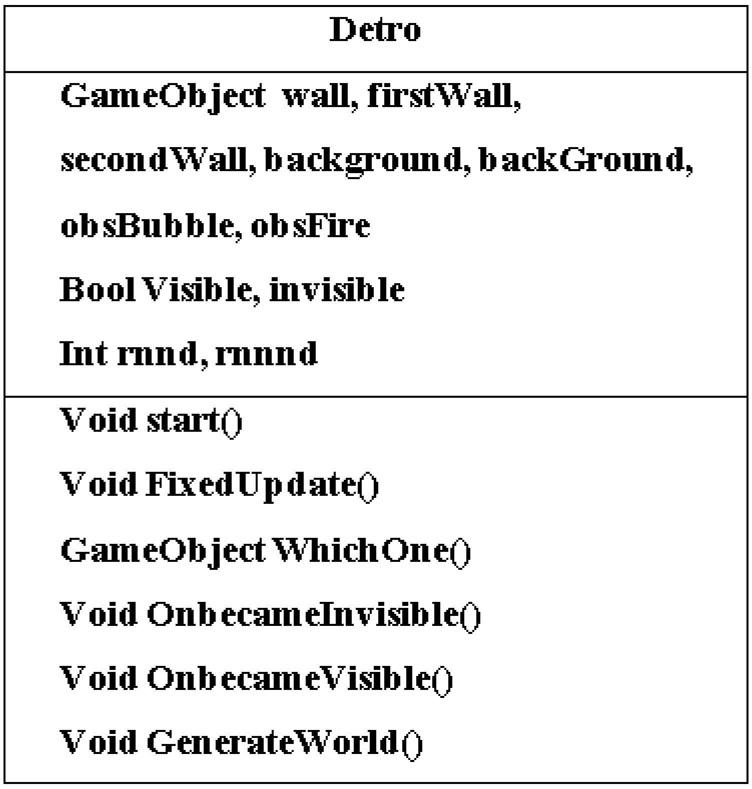


Figure 1.3- ViewSys class

At this stage we are creating class which will be responsible on the rectangles which are the key so the program will know with which character the player wish to interact with



```
Detro
-----------------------------------------
GameObject  wall, firstWall,
secondWall, background, backGround,
obsBubble, obsFire
Bool Visible, invisible
Int rnnd, rnnnd
-----------------------------------------
Void start()
Void FixedUpdate()
GameObject WhichOne()
Void OnbecameInvisible()
Void OnbecameVisible()
Void GenerateWorld()
```

Figure 1.4 - Destro class

This class is the most important because this is the one who is creating the world where the characters cn move and also is destroying what it builds by detecting if the objects created are visible or not, by this way the program do not use a lot of device memory and this by minimalizing the number of game object created or existing.

A class diagram is shown on following figure.

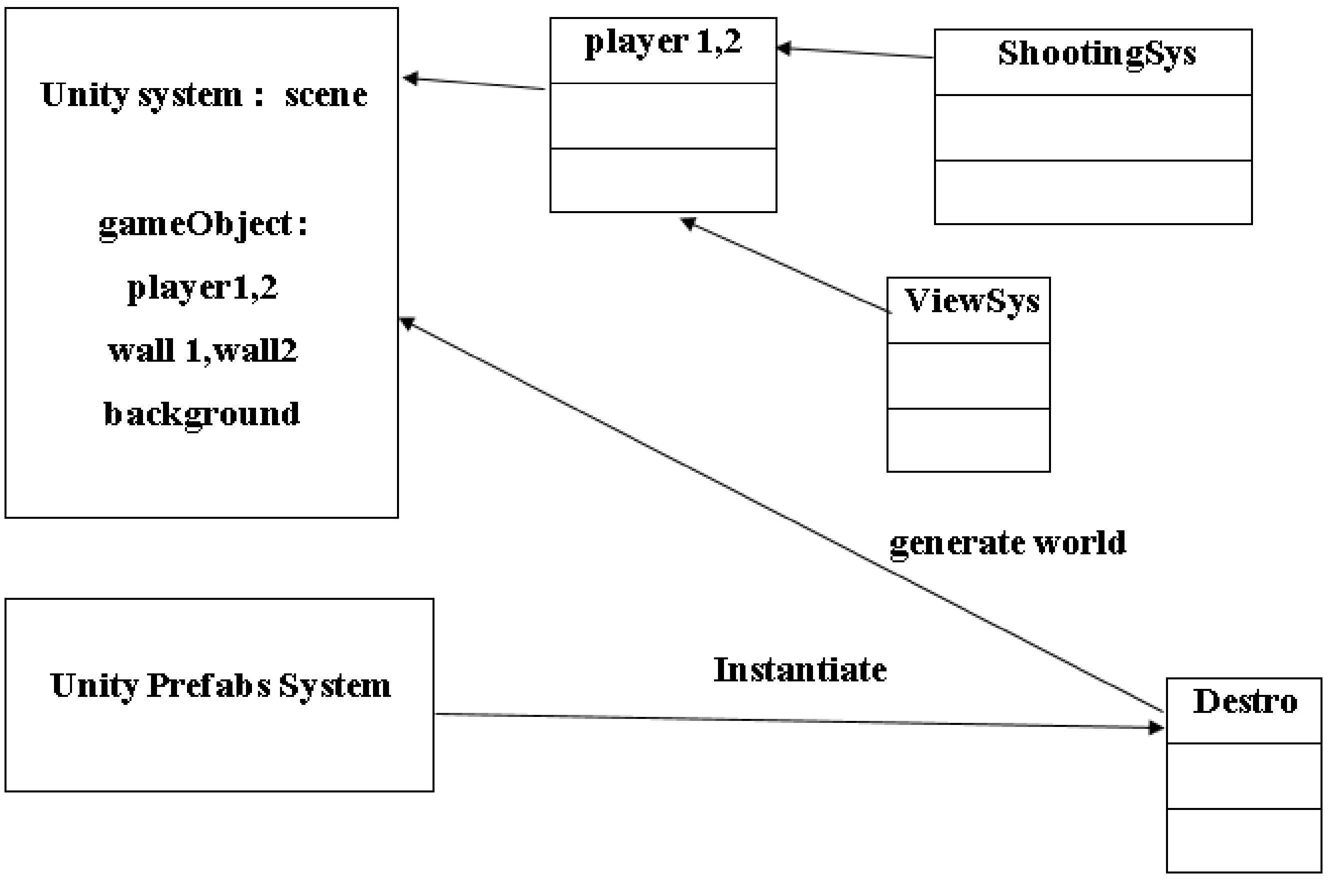


Figure 1.5 - Class diagram

Component diagram is shown on the following figure.

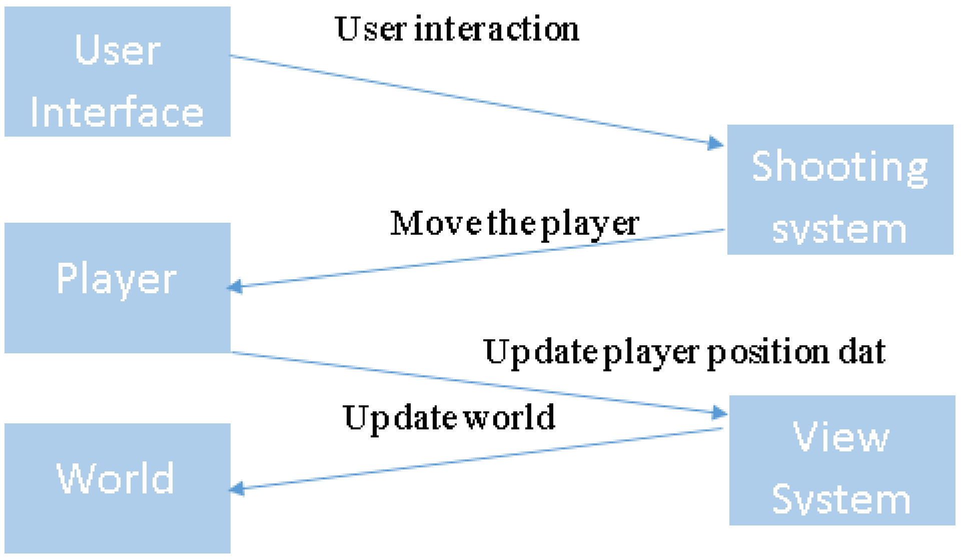


Figure 1.6 – Component diagram

1.3 Character design

On this project we are working with adobe animate for animation and also for the character design. The character is an alien who is based on cartoon style (Fig. 1.7).

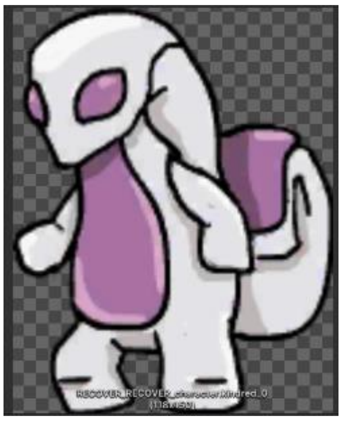


Figure 1.7- Character

What mean the character is divided on three equal part head body legs simple and based on just 2 colors white and purple, the character has 3 basic animations, first idle when he is just waiting, the second is flipping into a ball to bounce, the third is just the opposite of the second animation with an addition of grabbing

2 TEHNOLOGIES USED

2.1 Unity

Unity is a cross-platform game engine developed by Unity Technologies, where you can make a game for PC and turn it to a mobile game with some easy modifications, not only pc and mobile platforms but also unity allow us to support a variety of console and virtual reality platforms.

User interface.

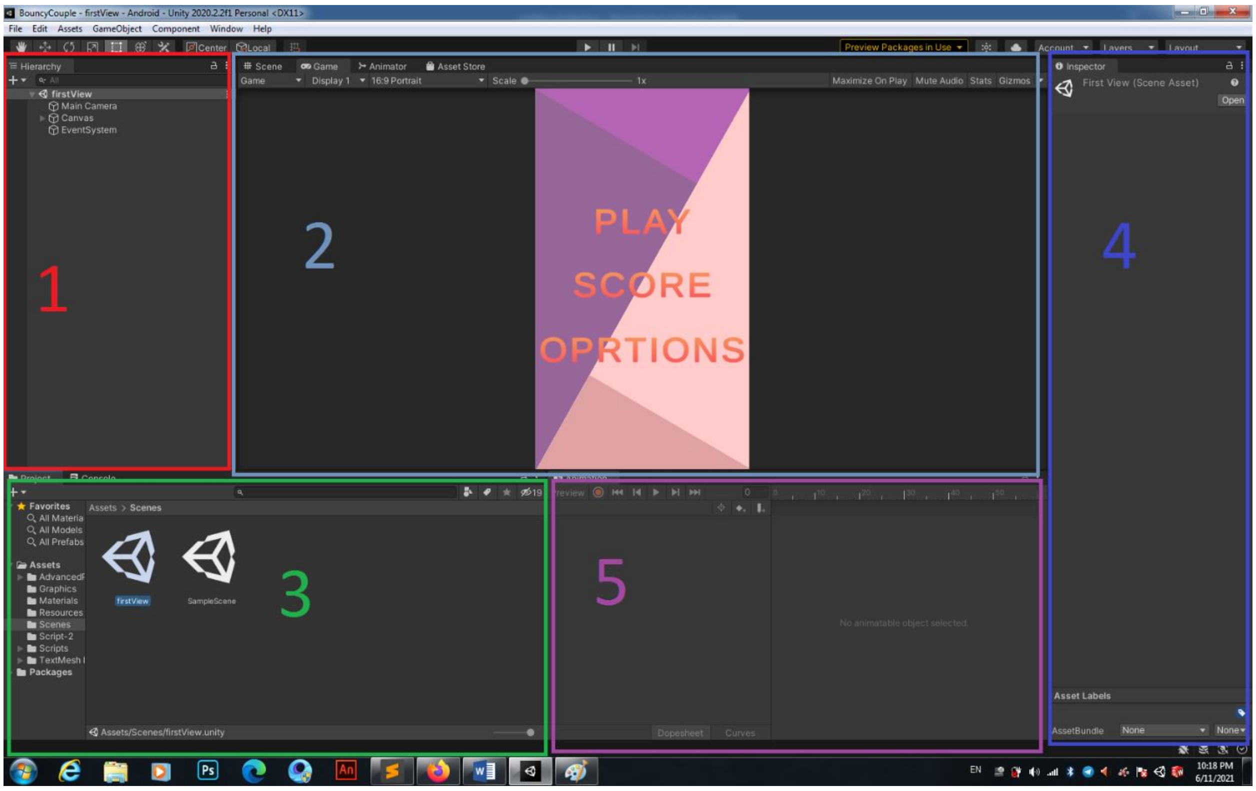


Figure 2.1 - Unity interface

At the first spot (red rectangle) we see the hierarchy where we can find all the game objects in a selected scene also where we can manually create delete game objects from the scene

The second spot (soft blue rectangle) we see the scene itself with 2 view the developer view and the game view, if we chose to work with the first view we can manipulate the game object manually and move, scale or rotate them easily in the second view we can’t interact with the game objects just as we define the interaction that should be in the game as the gamer.

The third spot (green rectangle) is the project window where we can find all the hierarchy of the project as folders, libraries, graphics and more this part let the developer search and find quickly any resources he need inside the project

The fourth spot (dark blue rectangle) the most important part which is the inspector, where we find all the proprieties that we need of any game object selected from the hierarchy of the scene, also we can add component and modify each component proprieties manually if it is not done by scripting.

The fifth spot (purple rectangle) the animation window where we do not make our animation but where we can adjust them and make them more fluid by manipulating the frames intervals and the orders.

The software architecture .

This was a view on the first and most important windows on a Unity platform there is more and more to discover and never get bored. Also I should to mention how Unity system actually wok, I mean by that just a big view not everything because the everything is too huge to explain; so basically Unity is based on one class called MonoBehaviour, this is the base class from which every Unity script inherit.

In this major class (as I like to call it the big mom) have some basic methods as:

Start(), Update(),Awake(), FixedUpdate(), OnCollisionEnter(), OnCollisionEnter2D(), OnMouseOver() and more

But what we can understand from all those methods is that all the manipulation we going to need in our project or any projects is already predefined in one class by Unity itself so if we read the documentation and more understood every practice of every method the world is in our hands cause we literary can create anything inside our small worlds (computers).

Also I should mention that unity make things more fluid by letting the developer use both scripting and manually techniques to get any idea built, so if we want use the manually methods to adjust for example a gameObject position we can select the gameObject from the scene hierarchy and edit the (x, y, z) values in the transform proprieties inside the inspector window, or also we can just with a simple line of code do so any time we would like or after an event that we program.

Talking about events.

Unity is too smart wanted to say the creators of Unity are too smart by creating an EventSystem inside the engine which make the development of games much easier, the EventSystem is simply a class where we can find all the methods and proprieties responsible to get an action triggered with the help of raycast, colliders and more and this by changing the states of any gameObject and add functionalities to those gameObject after we define how and when.

There is much more to talk about unity the collision system, physics system, animation system and the list closely never finish, I hope I can teach this material to get more deeply inside the world of unity, at least I'm going to be doing something I love.

2.2 Adobe animate

Here we go with another story this software is dedicated to drawing and animation more specifically 2D animation, know long time ago by flash also we can make simple games with it where you animate and script the animation.

The most known practice of animation process is frame by frame, but also with animate we can find the best the bones animation process where we make a character in parts (head, LeftArm, Righter, etc.) and assemble them by bones then make the position we would like to have in each sequence of time.

This practice of animation is known by Skeletal animation or rigging, and by this way we can make the work much easy and also done it more faster, unless if you are lazy more than me.

The software after finishing the animation make you the opportunity to export the work with different ways like SVG file or sprite sheet or a gif … the most possibilities you need you can for sure find it with adobe animate.

The program support both rigging animation and frame by frame animation, also it support the vectorization model so whatever shape you draw and however you zoomed in the work will still perfectly intact, also it support the pixel art but must download a package as an add-on.

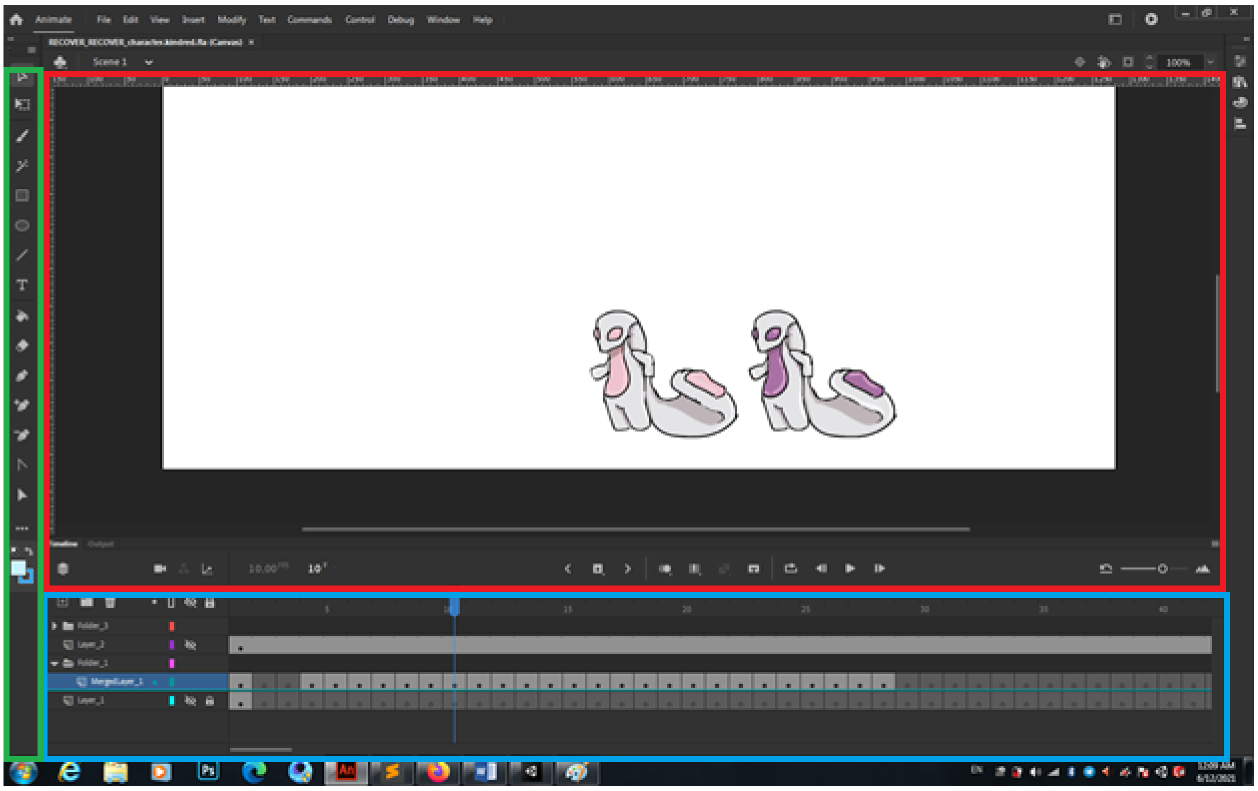


Figure 2.2 - Adobe animate interface

Let’s break apart the adobe animate window the green box we find all the tools we need from the brush tell the color-set.

The red box is the scene where we can see our creativity, also it is equipped with a rule that you can show or cancel.

The blue box is the most important which is the time line and here where ll the work is done we can find the layers and folders, also a menu where the onion, repetition zoom, camera tools. The interface actually is simple and clean.

I forgot to motion that adobe animate allow also the fact that you can animate by scripting, which give you much more control and smoothness at work, and to see the power of this software it support also the Skeletal animation process on 2D which give the artist much more time saving working on huge project as animated movies or games.

2.3 C-sharp

C# (pronounced see sharp) is a general-purpose, multi-paradigm programming language encompassing static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines

C# was developed around 2000 by Microsoft as part of its .NET initiative and later approved as an international standard by Ecma in 2002 and ISO in 2003. It was designed by Anders Hejlsberg, and its development team is currently led by Mads Torgersen, being one of the programming languages designed for the Common Language Infrastructure (CLI). The most recent version is 9.0, which was released in 2020 in .NET 5.0 and included in Visual Studio 2019 version 16.8

Mono is a free and open-source project to develop a cross-platform compiler and runtime environment (i.e. virtual machine) for the language.

The Ecma standard lists these design goals for C#

The language is intended to be a simple, modern, general-purpose, object-oriented programming language.

The language, and implementations thereof, should provide support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.

The language is intended for use in developing software components suitable for deployment in distributed environments.

Portability is very important for source code and programmers, especially those already familiar with C and C++.

Support for internationalization is very important.

C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.

Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language.

3 THE BUILDING PROCESS

3.1 Animation process

The animation process I used on this project is the frame by frame animation, the frame-by-frame animation changes the contents of the Stage in every frame. It is best suited to complex animation in which an image changes in every frame instead of simply moving across the Stage. Frame-by-frame animation increases file size more rapidly than twined animation. In frame-by-frame animation, Animate stores the values for each complete frame.

To create a frame-by-frame animation, define each frame as a keyframe and create a different image for each frame. Each new key frame initially contains the same contents as the key frame preceding it, so you can modify the frames in the animation incrementally.

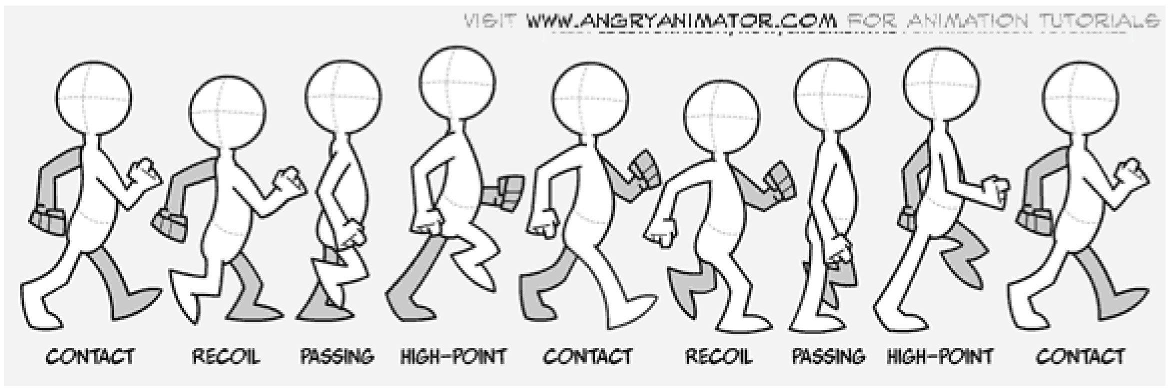


Figure 3.1 - Frame by frame animation 1

On Adobe Animate is simple to go forward and back on the time line using > and < keys, and to create a key frame we use F6 to add the in-between frames we use F5, after selecting the frame we which to work with we start drawing the position and go to the next frame.

Inbetweening, also commonly known as tweening, is a process in animation that involves generating intermediate frames, called in-betweens, between two key frames.

The intended result is to create the illusion of movement by smoothly transitioning one image into another.

Also we have the ease which give as much more smoothness to our animation even if the FPS (frame by second is high).

On this project we have 4 basics animation

Idle when the character is with zero interaction with the player with just 3 frames

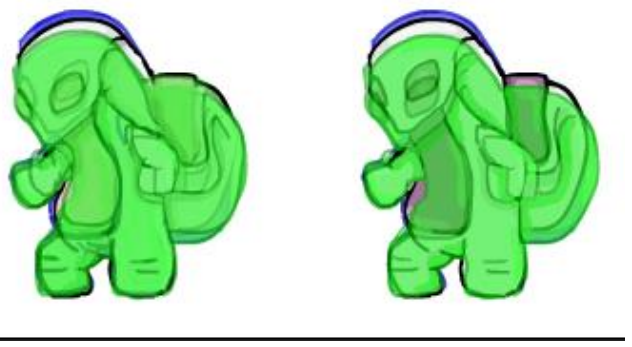


Figure 3.2 - frame by frame animation 2

The spin when the character is moving by bouncing between the walls with 18 frames

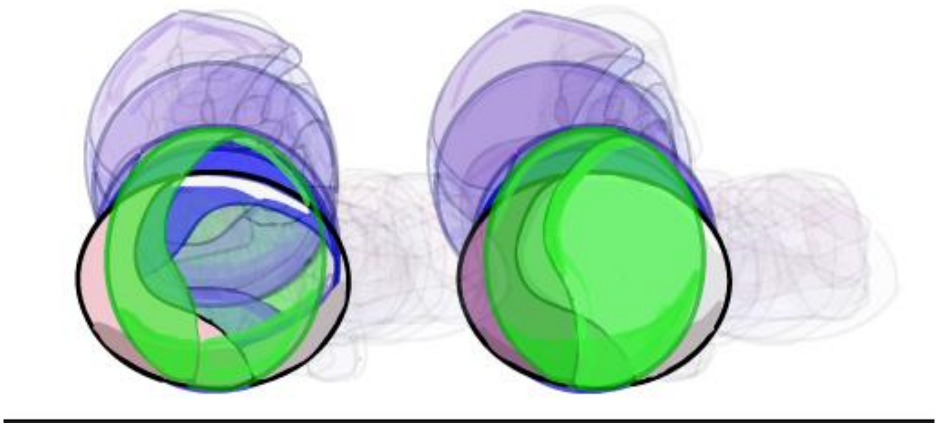


Figure 3.3 - Frame by frame animation 3

The release after the character stop moving and grab on to a wall with 7 frames

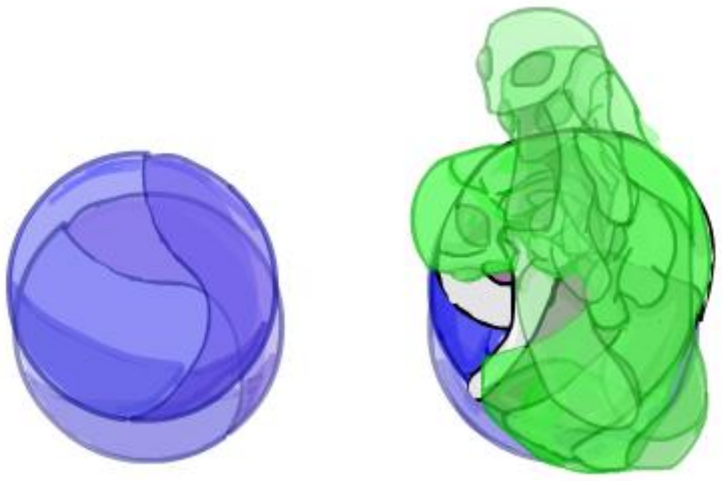


Figure 3.4 - frame by frame animation 4

The death when the character touch an obstacle and die and end the game.

The tools used for this animations is Adobe Animate and a graphical tablet named “Wacom”.

Let’s not forget that not just the character who need the animation also the obstacles, where we find three types of obstacle fire obstacles with flames

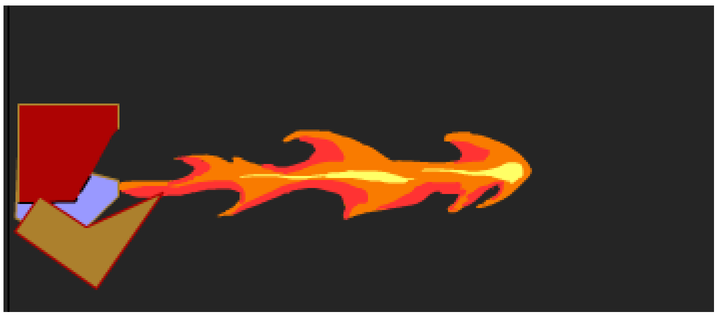


Figure 3.5 - flame obstacle

Also, the bubble obstacles.

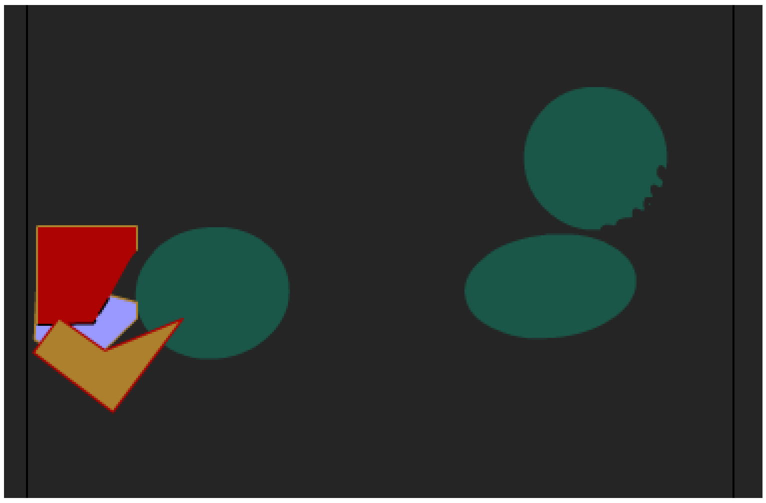


Figure 3.6- bubble obstacle

And finally the water which is the common obstacle between the two characters.

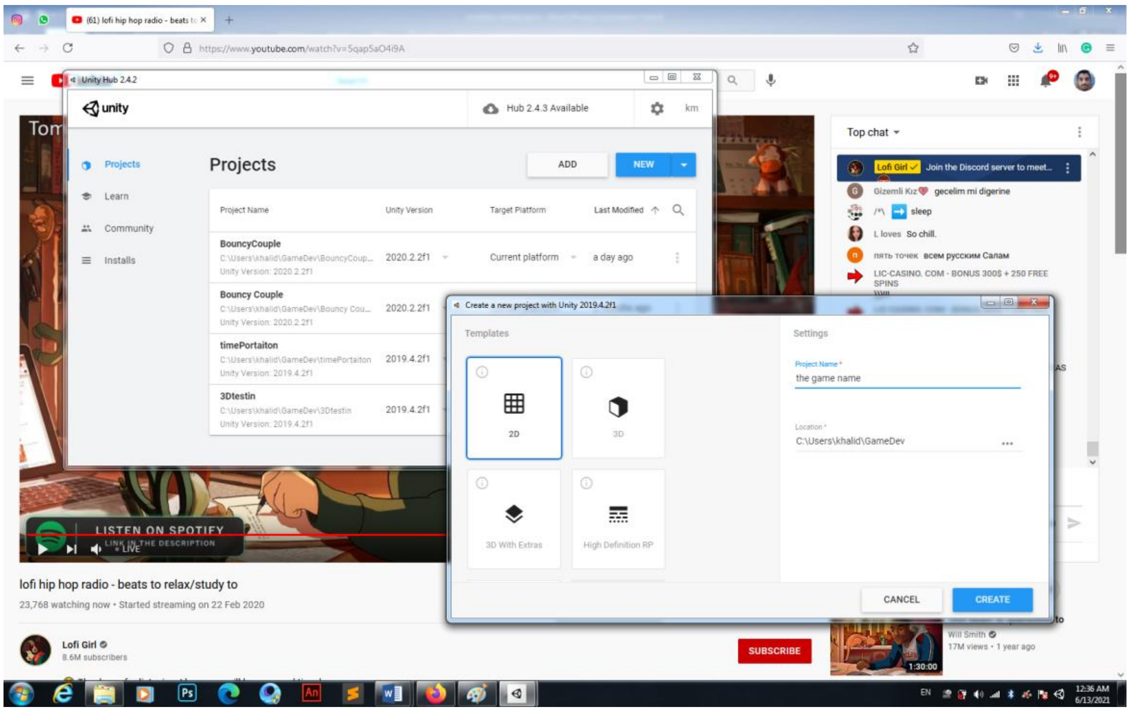
3.2 Unity scenes building

At the beginning we should start unity hub to create a new project, unity hub is a client application for unity where you cn check for the last updates and the profile and account settings.

After that we get a new project with the sample scene and with a camera as a game object.

In this project we are going to use two scenes one for the menu and the user interface and the second one for the game itself.

At the first scene the menu scene we find a camera a canvas which is the parent object of 3 menu the main menu the options menu and the high score menu.



Project Name	Unity Version	Target Platform	Last Modified
BouncyCouple C:\Users\khalid\GameDev\BouncyCoup... Unity Version: 2020.2.2f1	2020.2.2f1	Current platform	a day ago
Bouncy Couple C:\Users\khalid\GameDev\Bouncy Cou... Unity Version: 2020.2.2f1	2020.2.2f1		
timePortaiton C:\Users\khalid\GameDev\timePortaiton Unity Version: 2019.4.2f1	2019.4.2f1		
3Dtestin C:\Users\khalid\GameDev\3Dtestin Unity Version: 2019.4.2f1	2019.4.2f1		

Settings
Project Name *: the game name
Location *: C:\Users\khalid\GameDev

Figure 3.7 - New unity project

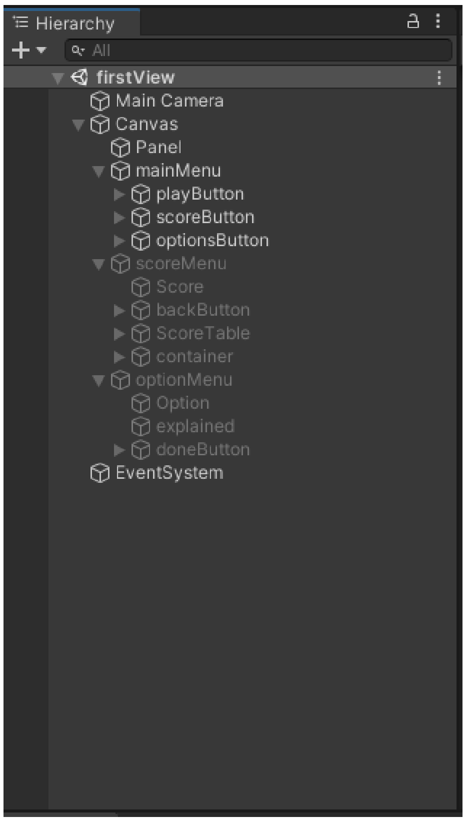

```
Hierarchy
▼ firstView
    Main Camera
  ▼ Canvas
      Panel
    ▼ mainMenu
      ▶ playButton
      ▶ scoreButton
      ▶ optionsButton
    ▼ scoreMenu
        Score
      ▶ backButton
      ▶ ScoreTable
      ▶ container
    ▼ optionMenu
        Option
        explained
      ▶ doneButton
    EventSystem
```

Figure 4.8 - Scene #1 hierarchy

And on the second scene we can find also a camera object but also the player object and n event System that help us to work with the line renderer to apply a movement after releasing on the game character with specific direction and speed and finally the start game Object that contain more subgmeObjects know s children parent relation (OOP) to create the first view of the world when starting.

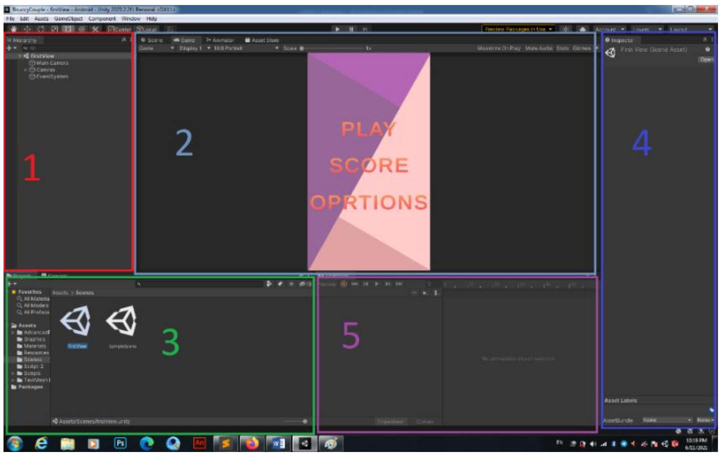
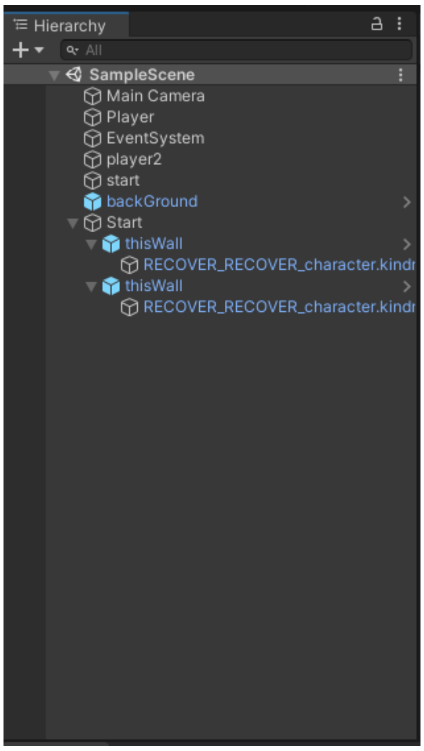


Figure 3.9 - Scene #1



```
Hierarchy
+ ▾  All
▾ SampleScene
    Main Camera
    Player
    EventSystem
    player2
    start
    backGround
  ▾ Start
    ▾ thisWall
        RECOVER_RECOVER_character.kindr
    ▾ thisWall
        RECOVER_RECOVER_character.kindr
```

Figure 3.10 - Scene#2 hierarchy

So, the scene looks like this (Fig. 3.11).

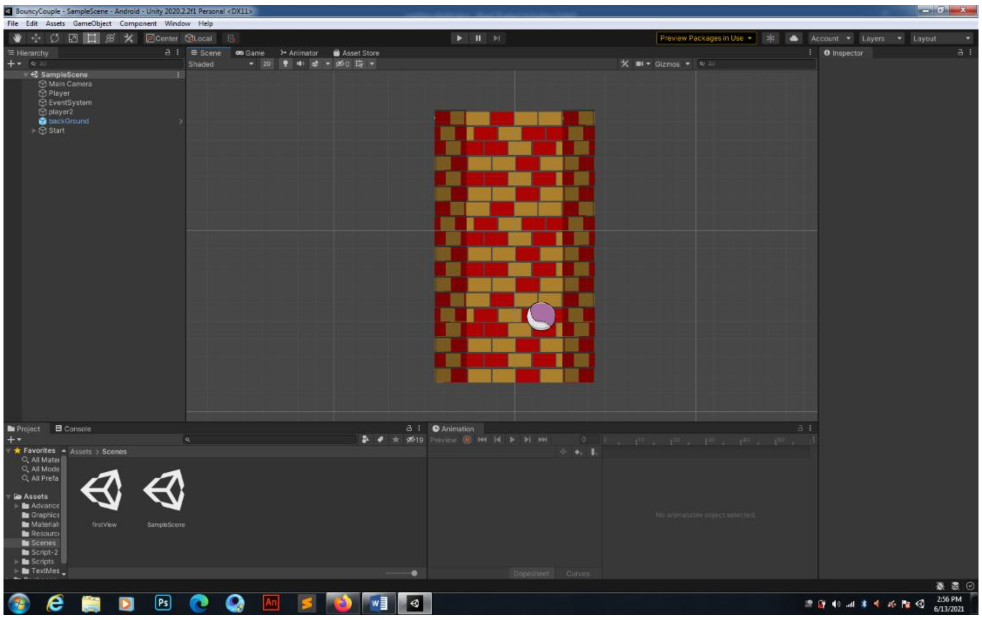


Figure 3.11 - Scene #2

After we got our scenes ready, we are going directly to the scripts, the new going to look on the relationship between them.

3.3 The scripts

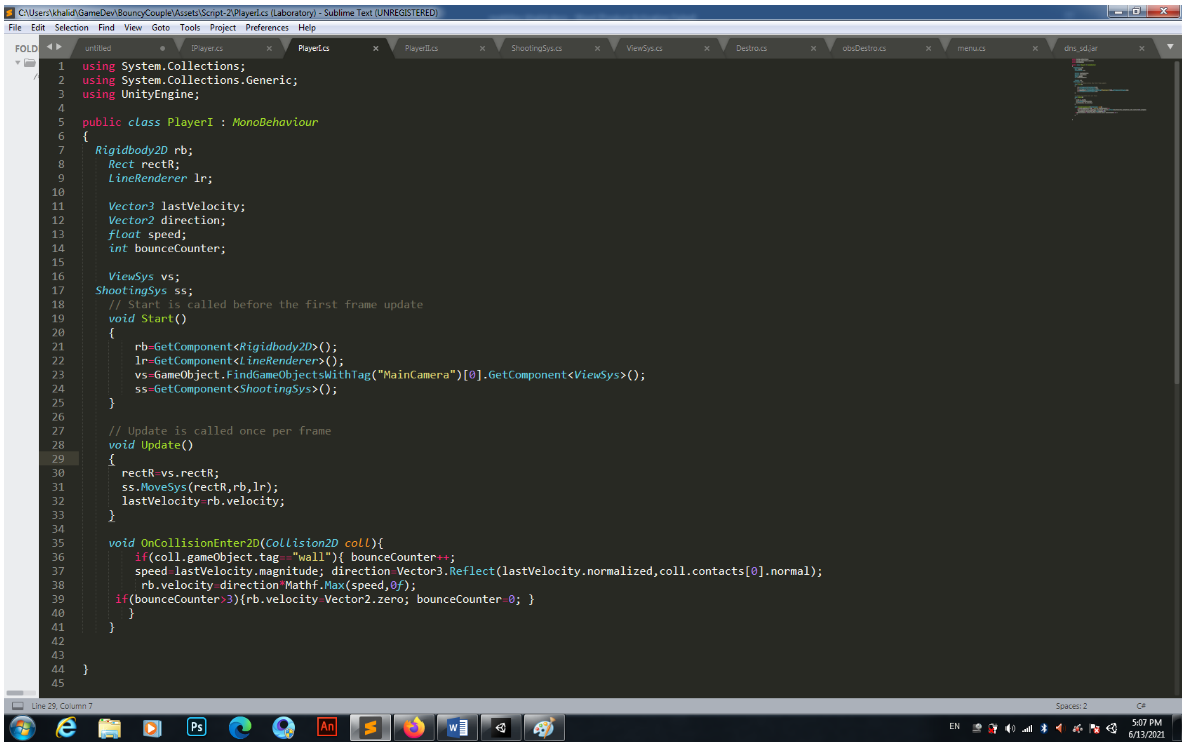
The magic with unity is you cn combine the manually work with the scripting work, so after we created the game objects we need manually we can attach them the C# script that is going to control his behavior.

We have basically seven scripts that control the movement of the players also the world generation and destroying, and finally the menu script.

Let's start by the first script **PlayerI.cs**At this stage of script as shown in figure 5, we have all the player controls trigger and the collision system, the movement is a method instantiated from another class **ShootingSys.cs.**

The method is named **MoveSys(Rect rectR, Rigidbody2D rb, LineRenderer lr)**

It support three parameters rectR which is a rectangle where we cn detect the user interaction with the first character, rb a Rigidbody2D who allow us to control the character itself by affecting his physics parameters, and finally a LineRenderer lr which is supposed to control when and where the shooting line is supposed to appear and disappear.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerI : MonoBehaviour
{
  Rigidbody2D rb;
    Rect rectR;
    LineRenderer lr;

    Vector3 lastVelocity;
    Vector2 direction;
    float speed;
    int bounceCounter;

    ViewSys vs;
  ShootingSys ss;
    // Start is called before the first frame update
    void Start()
    {
        rb=GetComponent<Rigidbody2D>();
        lr=GetComponent<LineRenderer>();
        vs=GameObject.FindGameObjectsWithTag("MainCamera")[0].GetComponent<ViewSys>();
        ss=GetComponent<ShootingSys>();
    }

    // Update is called once per frame
    void Update()
    {
      rectR=vs.rectR;
      ss.MoveSys(rectR,rb,lr);
      lastVelocity=rb.velocity;
    }

    void OnCollisionEnter2D(Collision2D coll){
        if(coll.gameObject.tag=="wall"){ bounceCounter++;
        speed=lastVelocity.magnitude; direction=Vector3.Reflect(lastVelocity.normalized,coll.contacts[0].normal);
         rb.velocity=direction*Mathf.Max(speed,0f);
     if(bounceCounter>3){rb.velocity=Vector2.zero; bounceCounter=0; }
       }
    }


}
```

Figure 3.12 - PlyerI.cs

also we have another part of movement system which is activated after detecting a collision with a gmeObject tagged ‘wall’, the algorithm responsible about this system is below:

```
       void OnCollisionEnter2D(Collision2D coll){
             if(coll.gameObject.tag=="wall"){ bounceCounter++;
             speed=lastVelocity.magnitude;
direction=Vector3.Reflect(lastVelocity.normalized,coll.contacts[0].normal
);
              rb.velocity=direction*Mathf.Max(speed,0f);
          if(bounceCounter>3){rb.velocity=Vector2.zero; bounceCounter=0;
}
            }     }
```

The method OnColisionEnter2D is a defined method at the MonoBehaviour,this method is activated when an incoming collider makes contact with this object's collider with 2D physics only, for the 3D process we have the same system but with another method named OnCollisionEnter.

Further information about the collision is reported in the Collision2D parameter passed during the call. If you don't need this information then you can declare OnCollisionEnter2D without the parameter.

At this part of the script (.class), we act just if the coll parameter which is the gmeObject that we are entering the collision with is tagged by the name “wall”.

Then we use three variables bounceCounter, speed and direction,

```
     /*  speed=lastVelocity.magnitude;
direction=Vector3.Reflect(lastVelocity.normalized,coll.contacts[0].normal
);
              rb.velocity=direction*Mathf.Max(speed,0f); */
```

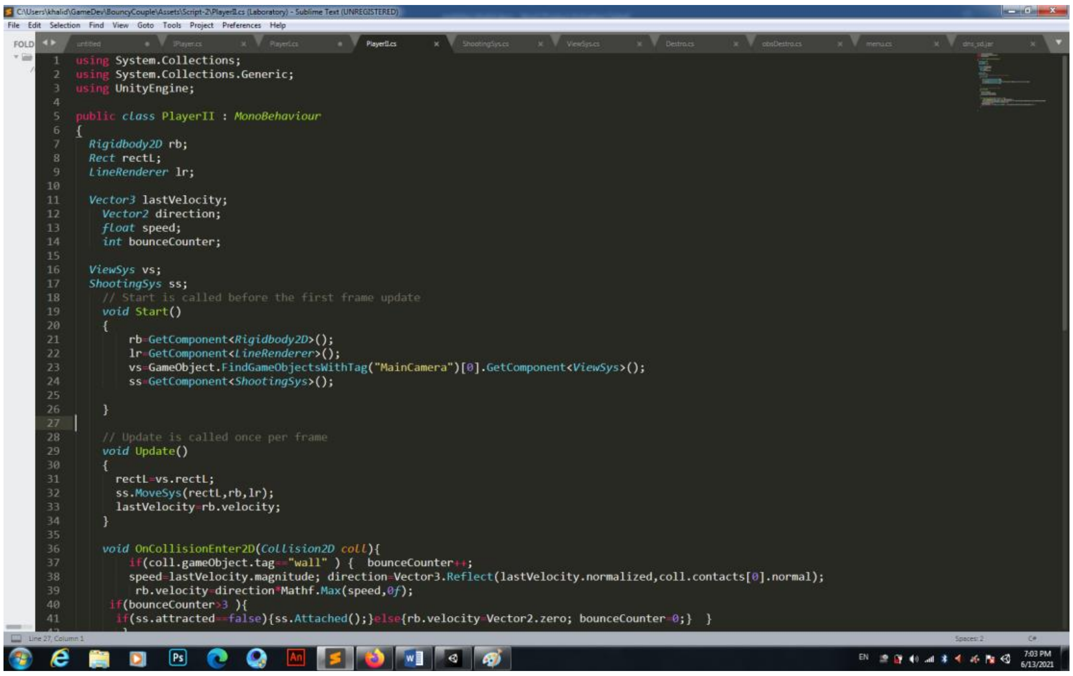
```
     the last two is to affect the rigidbody2D velocity and the first
variable is to stop the velocity after three bouncing.
```

```
     /* bounceCounter++;
     if(bounceCounter>3){rb.velocity=Vector2.zero; bounceCounter=0; }*/
```

and about the two first methods Start() and Update() there also predefined in the MonoBehaviour class, I guess you know why I call it earlier the big mom, the start method is created to initialize all the attributes of the class and if an initialization is needed before this method we still can use the awake() method which is similar but work earlier,

Also we have the update method so this one is much more important because this is unity this where the unity engine is calling this method every frame to check for the updates at the scene and characters stats, also we can use another method for the same purpose named FixedUpdate() which is much more precise where the engine call this method not after every frame but after a fixed time that we can set manually on unity interface, and it is known that is better for physics updates.

After we took a look on the playerI.cs the playerII.cs is the same just with one modification for the rigidbody affected and the rectangle that is going to inform the direction of the movement.



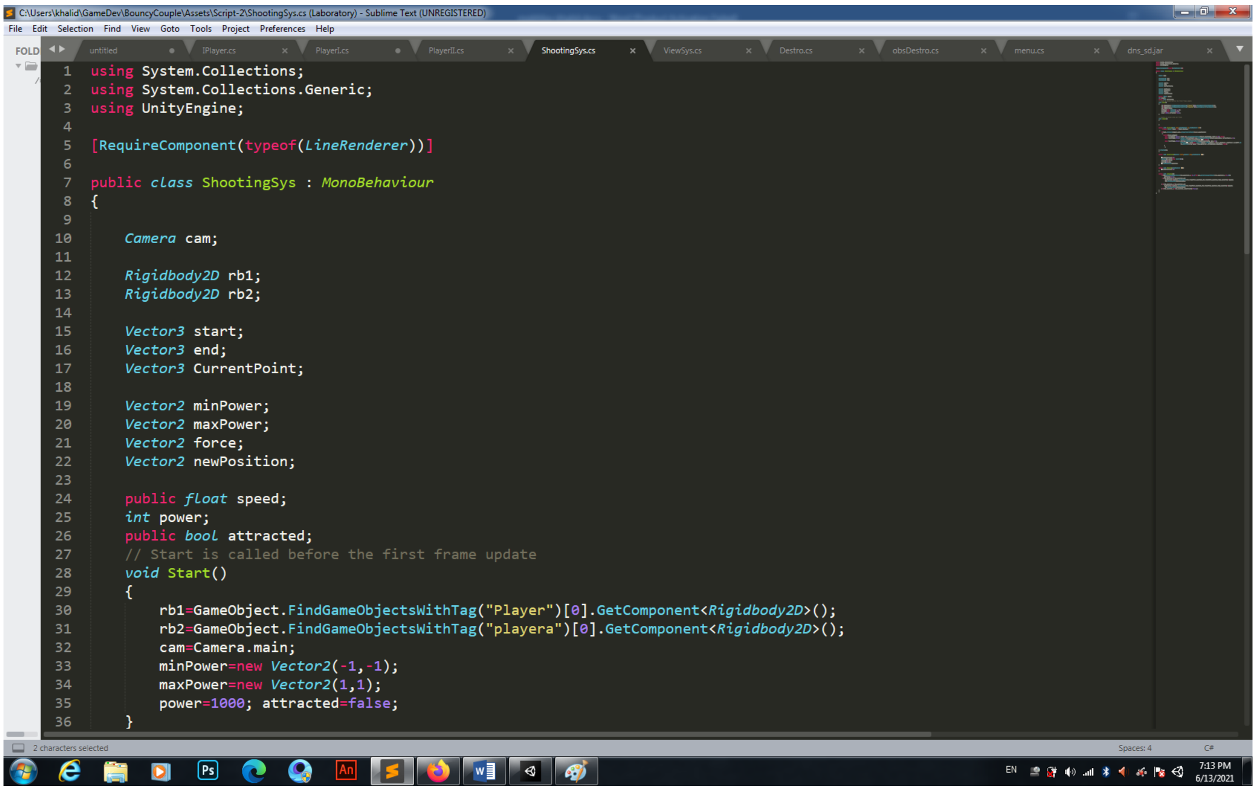
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerII : MonoBehaviour
{
  Rigidbody2D rb;
  Rect rectL;
  LineRenderer lr;

  Vector3 lastVelocity;
    Vector2 direction;
    float speed;
    int bounceCounter;

  ViewSys vs;
  ShootingSys ss;
    // Start is called before the first frame update
    void Start()
    {
        rb=GetComponent<Rigidbody2D>();
        lr=GetComponent<LineRenderer>();
        vs=GameObject.FindGameObjectsWithTag("MainCamera")[0].GetComponent<ViewSys>();
        ss=GetComponent<ShootingSys>();

    }

    // Update is called once per frame
    void Update()
    {
      rectL=vs.rectL;
      ss.MoveSys(rectL,rb,lr);
      lastVelocity=rb.velocity;
    }

    void OnCollisionEnter2D(Collision2D coll){
        if(coll.gameObject.tag=="wall" ) {  bounceCounter++;
        speed=lastVelocity.magnitude; direction=Vector3.Reflect(lastVelocity.normalized,coll.contacts[0].normal);
         rb.velocity=direction*Mathf.Max(speed,0f);
     if(bounceCounter>3 ){
      if(ss.attracted==false){ss.Attached();}else{rb.velocity=Vector2.zero; bounceCounter=0;}  }
```

Figure 3.13 - PlayerII.cs

While I tried to practice the oriented object programing and encapsulate the player class 1and 2 classes in an interface named IPlayer.cs but when I did it after shooting from one rectangle both the characters move, what was not planed at all.

Also we have the **ShootingSys.cs** script that have the moveSys method defined also we have a lot of methods like RenderLine() and EndLine() which are responsible on the shooting line, and the last method Attached() which let the two characters not separated and still appear in one screen view.

```
public void MoveSys(Rect rect,Rigidbody2D rb,LineRenderer lr){
        foreach (Touch touch in Input.touches)
      {
              switch(touch.phase){
                case TouchPhase.Began:
                case TouchPhase.Moved:
                case TouchPhase.Ended:
                 } }
```

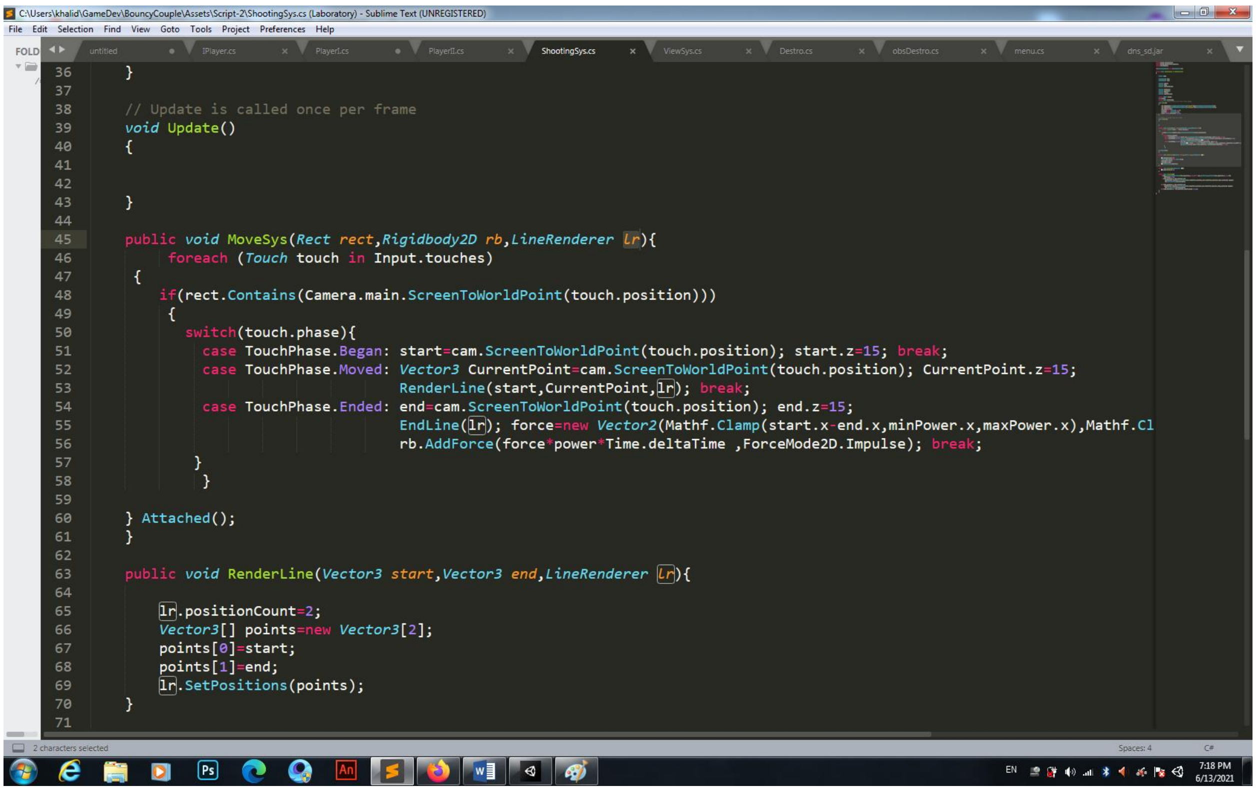

C:\Users\khalid\GameDev\BouncyCouple\Assets\Script-2\ShootingSys.cs (Laboratory) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
untitled
IPlayer.cs
PlayerI.cs
PlayerII.cs
ShootingSys.cs
ViewSys.cs
Destro.cs
obsDestro.cs
menu.cs
dns_sd.jar
2 characters selected
Spaces: 4
C#

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(LineRenderer))]

public class ShootingSys : MonoBehaviour
{

    Camera cam;

    Rigidbody2D rb1;
    Rigidbody2D rb2;

    Vector3 start;
    Vector3 end;
    Vector3 CurrentPoint;

    Vector2 minPower;
    Vector2 maxPower;
    Vector2 force;
    Vector2 newPosition;

    public float speed;
    int power;
    public bool attracted;
    // Start is called before the first frame update
    void Start()
    {
        rb1=GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<Rigidbody2D>();
        rb2=GameObject.FindGameObjectsWithTag("playera")[0].GetComponent<Rigidbody2D>();
        cam=Camera.main;
        minPower=new Vector2(-1,-1);
        maxPower=new Vector2(1,1);
        power=1000; attracted=false;
    }
```

Figure 3.14 - ShootingSys.cs part#1

This part of the code is specified to manage the screen touch interactions, where we can discover the touch.phase enum type that contain three touch phases which are began at the beginning of the contact, moved is while the contact is still true but moving modifying the x and y values, and ended is when the contact is dead and there is no more thouching.

And let’s do not forget that this system is capable to manage ten different touching interactions once at the same time.



C:\Users\khalid\GameDev\BouncyCouple\Assets\Script-2\ShootingSys.cs (Laboratory) - Sublime Text (UNREGISTERED)

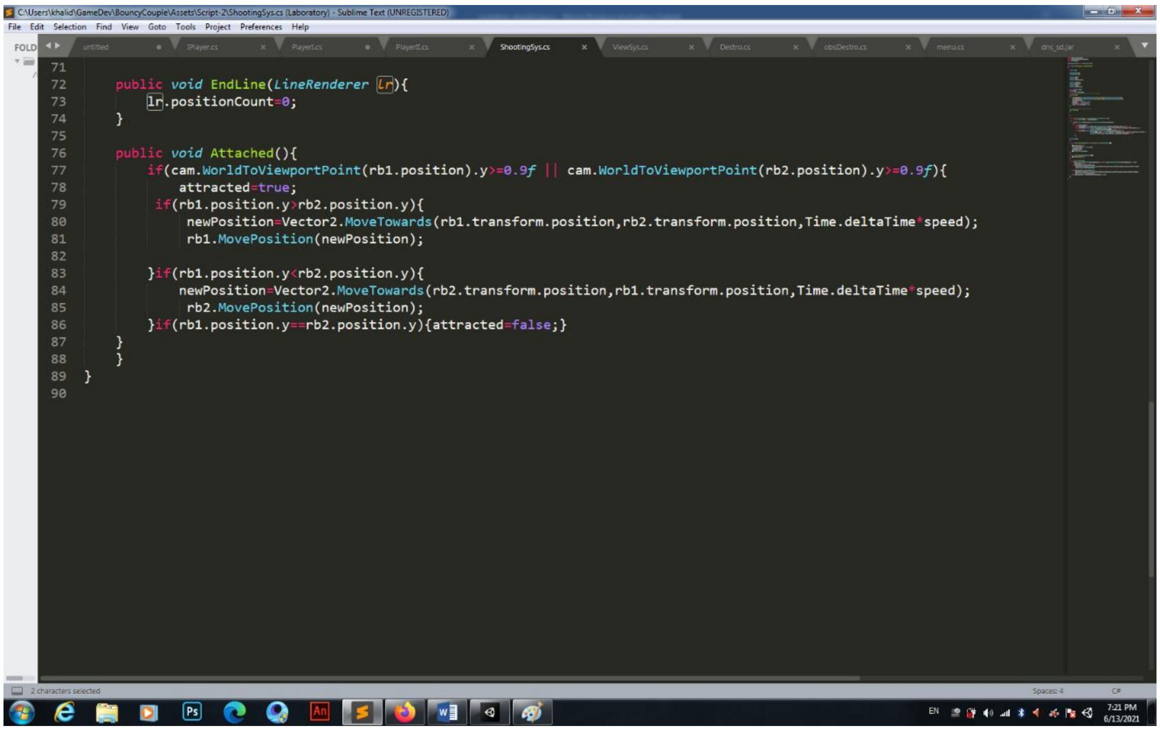
```
    }

    // Update is called once per frame
    void Update()
    {


    }

    public void MoveSys(Rect rect,Rigidbody2D rb,LineRenderer lr){
         foreach (Touch touch in Input.touches)
     {
        if(rect.Contains(Camera.main.ScreenToWorldPoint(touch.position)))
         {
           switch(touch.phase){
             case TouchPhase.Began: start=cam.ScreenToWorldPoint(touch.position); start.z=15; break;
             case TouchPhase.Moved: Vector3 CurrentPoint=cam.ScreenToWorldPoint(touch.position); CurrentPoint.z=15;
                                    RenderLine(start,CurrentPoint,lr); break;
             case TouchPhase.Ended: end=cam.ScreenToWorldPoint(touch.position); end.z=15;
                                    EndLine(lr); force=new Vector2(Mathf.Clamp(start.x-end.x,minPower.x,maxPower.x),Mathf.Cl
                                    rb.AddForce(force*power*Time.deltaTime ,ForceMode2D.Impulse); break;
             }
              }

    } Attached();
    }

    public void RenderLine(Vector3 start,Vector3 end,LineRenderer lr){

        lr.positionCount=2;
        Vector3[] points=new Vector3[2];
        points[0]=start;
        points[1]=end;
        lr.SetPositions(points);
    }
```

Figure 3.15 -1 ShootingSys.cs part#2

And the RenderLine() method is a simple algorithm that help us just to draw the line when the contact between the user's finger and the screen is done and ended and this by using the lineRenderer component already created on booth players gameObject.

As we can see some of the methods that we can apply on that component is SetPosition(Vector3[] points) also one of the most main attribute of this component is the positionCount which allow us to set how many dote we would like to draw after we know each dote position and storing them in the points Vector3 array.

After that of course who said render a line said end line which is the next method and it is the one who allow us to make the line rendered before end existing and that after the TouchPhase.Ended which means that the interaction between the finger touch is dead.



```
    public void EndLine(LineRenderer lr){
        lr.positionCount=0;
    }

    public void Attached(){
        if(cam.WorldToViewportPoint(rb1.position).y>=0.9f || cam.WorldToViewportPoint(rb2.position).y>=0.9f){
            attracted=true;
         if(rb1.position.y>rb2.position.y){
             newPosition=Vector2.MoveTowards(rb1.transform.position,rb2.transform.position,Time.deltaTime*speed);
             rb1.MovePosition(newPosition);

        }if(rb1.position.y<rb2.position.y){
            newPosition=Vector2.MoveTowards(rb2.transform.position,rb1.transform.position,Time.deltaTime*speed);
             rb2.MovePosition(newPosition);
        }if(rb1.position.y==rb2.position.y){attracted=false;}
    }
    }
}
```

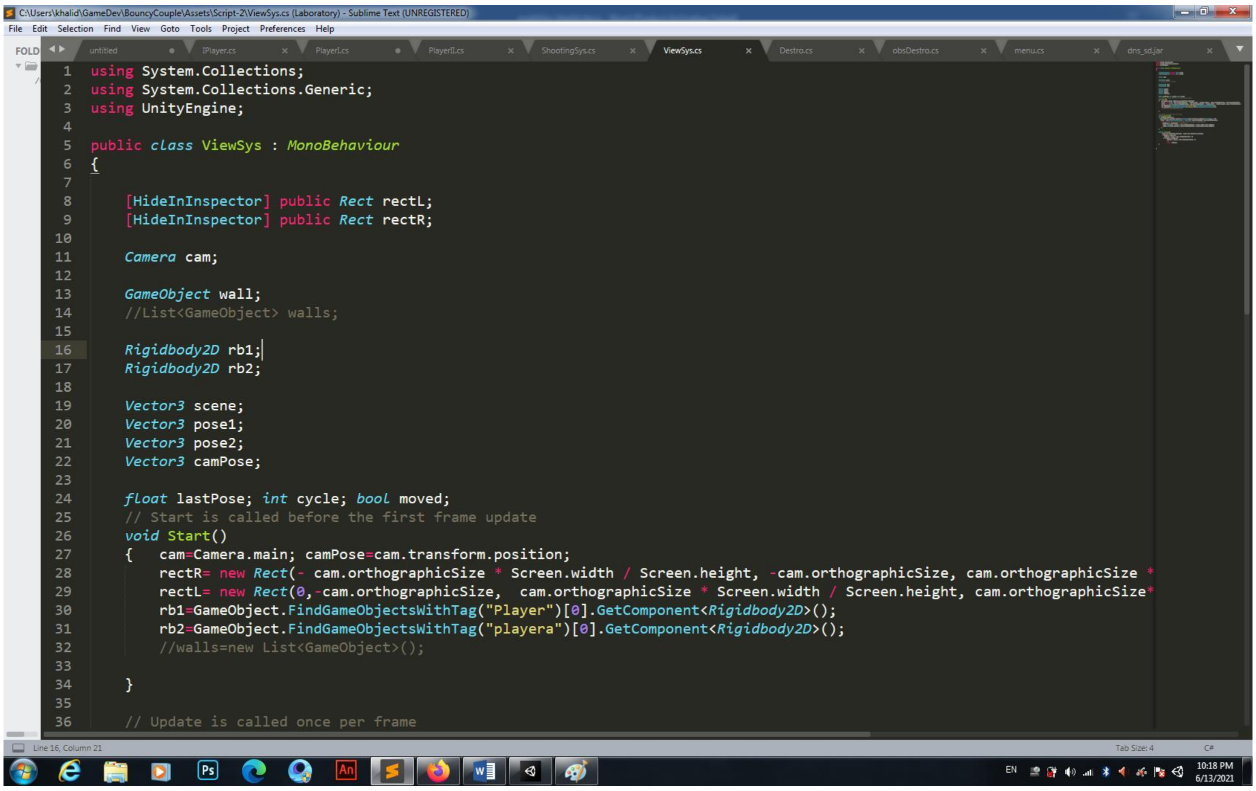
Figure 2 - ShootingSys.cs part#3

And finally with this script we find attached method that I created to control the movement of the second character while the first character is always down the screen view the other player cannot cross the top of the screen view.

And for this I get the help of two major predefined methods WorldToViewportPoint and MoveTowards, the first one is translating the position of a visible gameObject from world position scale it to camera view scale, where we can find that The bottom-left of the screen is (0,0); bottom-right of the screen is (1,0), top-left is (0, 1) and the top-right is (1,1). The center point will be (.5f, .5f), and by knowing the camera position it is easy to convert the world scale to the camera scale, this method take as parameter a vector3 which is the gmeObject we need position.

The second method is MoveTowards this method take three parameters at once, first the start position a vector3, the destination position also vetor3, and finally a float which is the time that must be done in the trajectory. It is a method that describe the movement then we can affect that movement to any rigidbody2d or even 3d by using the third method MovePosition and giving the movement created as a parameter.

Now we are going to take a look at **ViewSys.cs** script:



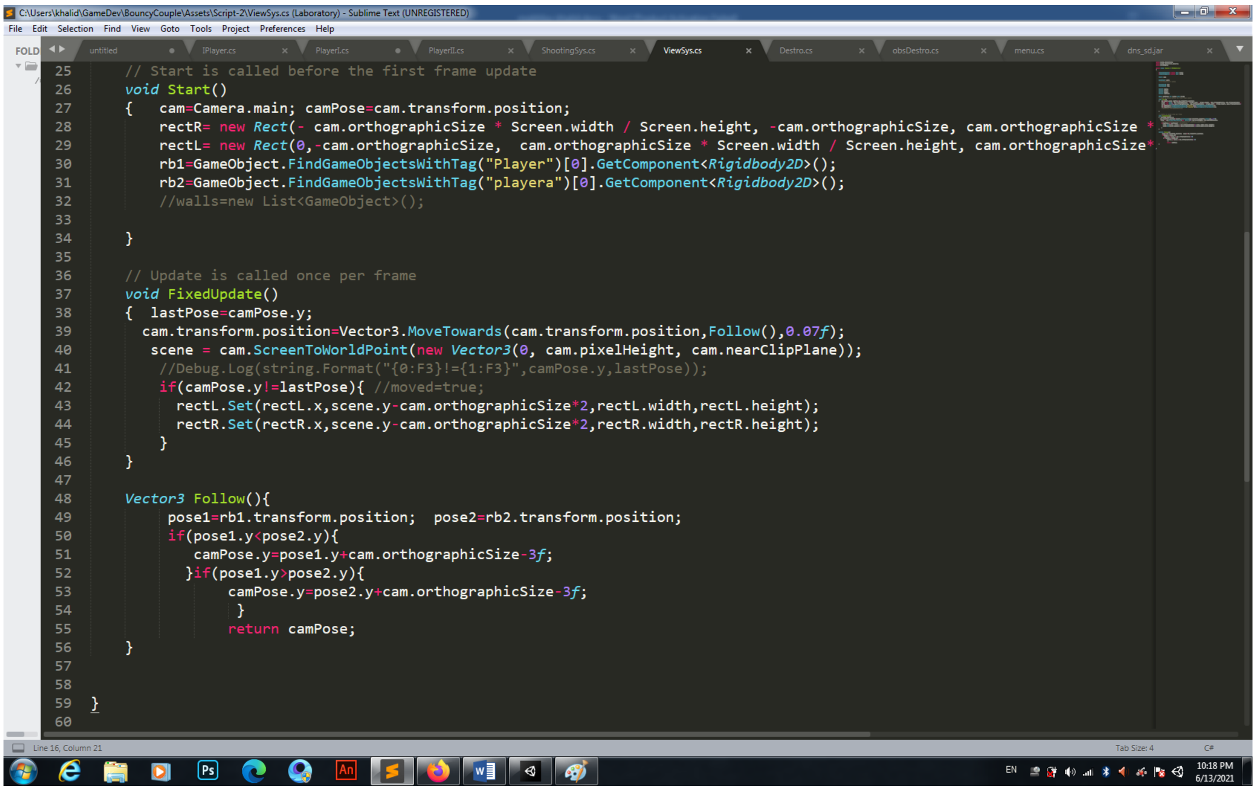
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ViewSys : MonoBehaviour
{

    [HideInInspector] public Rect rectL;
    [HideInInspector] public Rect rectR;

    Camera cam;

    GameObject wall;
    //List<GameObject> walls;

    Rigidbody2D rb1;
    Rigidbody2D rb2;

    Vector3 scene;
    Vector3 pose1;
    Vector3 pose2;
    Vector3 camPose;

    float lastPose; int cycle; bool moved;
    // Start is called before the first frame update
    void Start()
    {   cam=Camera.main; camPose=cam.transform.position;
        rectR= new Rect(- cam.orthographicSize * Screen.width / Screen.height, -cam.orthographicSize, cam.orthographicSize *
        rectL= new Rect(0,-cam.orthographicSize,  cam.orthographicSize * Screen.width / Screen.height, cam.orthographicSize*
        rb1=GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<Rigidbody2D>();
        rb2=GameObject.FindGameObjectsWithTag("playera")[0].GetComponent<Rigidbody2D>();
        //walls=new List<GameObject>();

    }

    // Update is called once per frame
```

Figure 3.17 - ViewSys prt#1

At this stage we are controlling the generation of the world and the obstacle also we are managing the creation and movement of two rectangles that separate the interaction between the user and the characters.

By using two attribute Rect rectL and rectR and hiding them in the inspector so we cant modifying them otherwise just by scripting.

The FixedUpdate method that we talked before is the one responsible of adjusting the position of the rectangles, and that by detecting if the camera position is adjusted and what can't be done just is one of the characters are changing their positions, I managed to control the camera movement by the second method named **Follow()** which test every fixed frame rate if the character 1 is above the second character or the opposite so the camera is going to follow always the one in the lower y position, so if the player moves the camera follow him then the rectangles positions are adjusted.



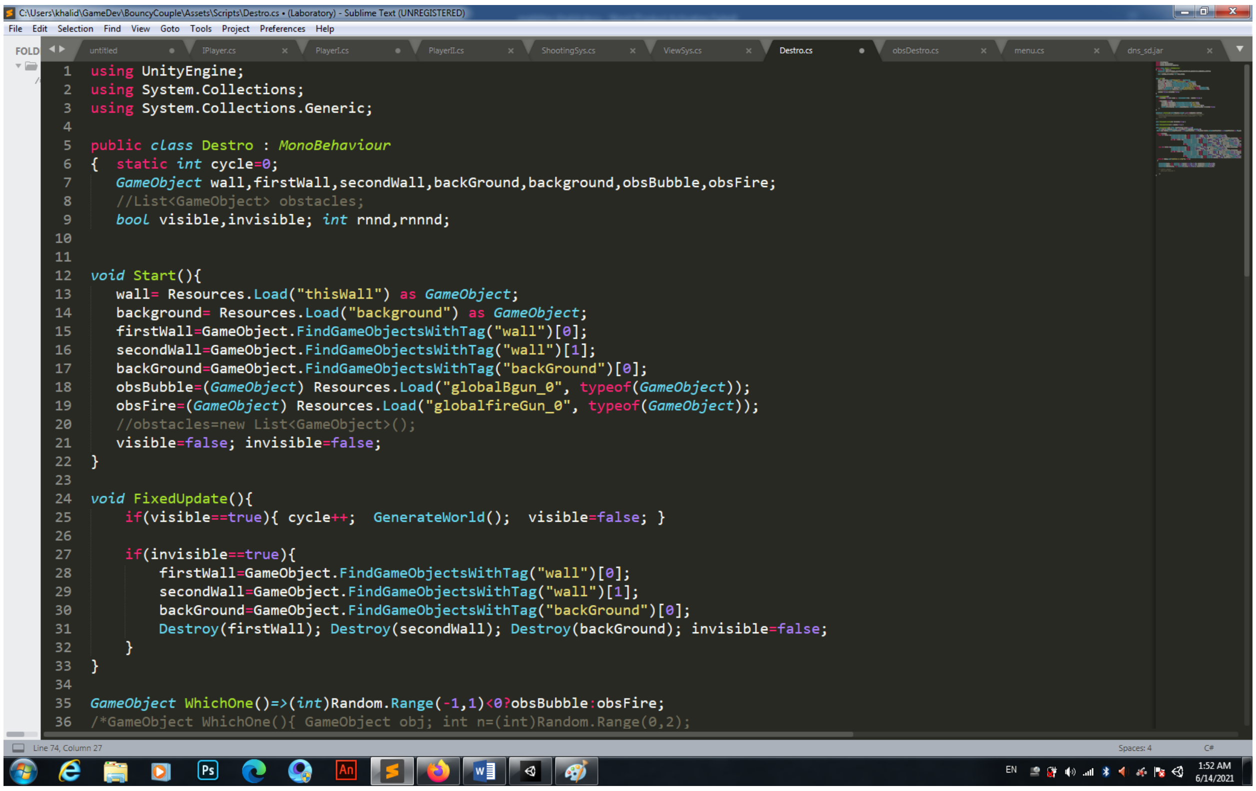
```
    // Start is called before the first frame update
    void Start()
    {   cam=Camera.main; camPose=cam.transform.position;
        rectR= new Rect(- cam.orthographicSize * Screen.width / Screen.height, -cam.orthographicSize, cam.orthographicSize *
        rectL= new Rect(0,-cam.orthographicSize,  cam.orthographicSize * Screen.width / Screen.height, cam.orthographicSize*
        rb1=GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<Rigidbody2D>();
        rb2=GameObject.FindGameObjectsWithTag("playera")[0].GetComponent<Rigidbody2D>();
        //walls=new List<GameObject>();

    }

    // Update is called once per frame
    void FixedUpdate()
    {  lastPose=camPose.y;
      cam.transform.position=Vector3.MoveTowards(cam.transform.position,Follow(),0.07f);
       scene = cam.ScreenToWorldPoint(new Vector3(0, cam.pixelHeight, cam.nearClipPlane));
        //Debug.Log(string.Format("{0:F3}!={1:F3}",camPose.y,lastPose));
        if(camPose.y!=lastPose){ //moved=true;
          rectL.Set(rectL.x,scene.y-cam.orthographicSize*2,rectL.width,rectL.height);
          rectR.Set(rectR.x,scene.y-cam.orthographicSize*2,rectR.width,rectR.height);
        }
    }

    Vector3 Follow(){
         pose1=rb1.transform.position;  pose2=rb2.transform.position;
         if(pose1.y<pose2.y){
            camPose.y=pose1.y+cam.orthographicSize-3f;
           }if(pose1.y>pose2.y){
                camPose.y=pose2.y+cam.orthographicSize-3f;
                 }
                return camPose;
    }


}
```

Figure 3.18 - ViewSys prt#2

When talking about the generation of world and obstacles we have **Destro.cs** script as shown bellow

This part at the FixedUpdate in this script is the responsible of generating the world and it is based on two variables **visible** and **invisible,** those variables are set under two other predefined methods from the big mom (MonoBehaviour).

The two methods are **OnBecameInvisible()** and **OnBecameVisible()**, the script is attached basically to a prefab I’m going to introduce you this term later but keep in mind that a prefab is a gameObject that we set the component and make it as a predefined GameObject which we can use by the script later on, so those two methods are activated after the gameObject attached to this script appear or disappear from the scene exactly all what the camera can project.

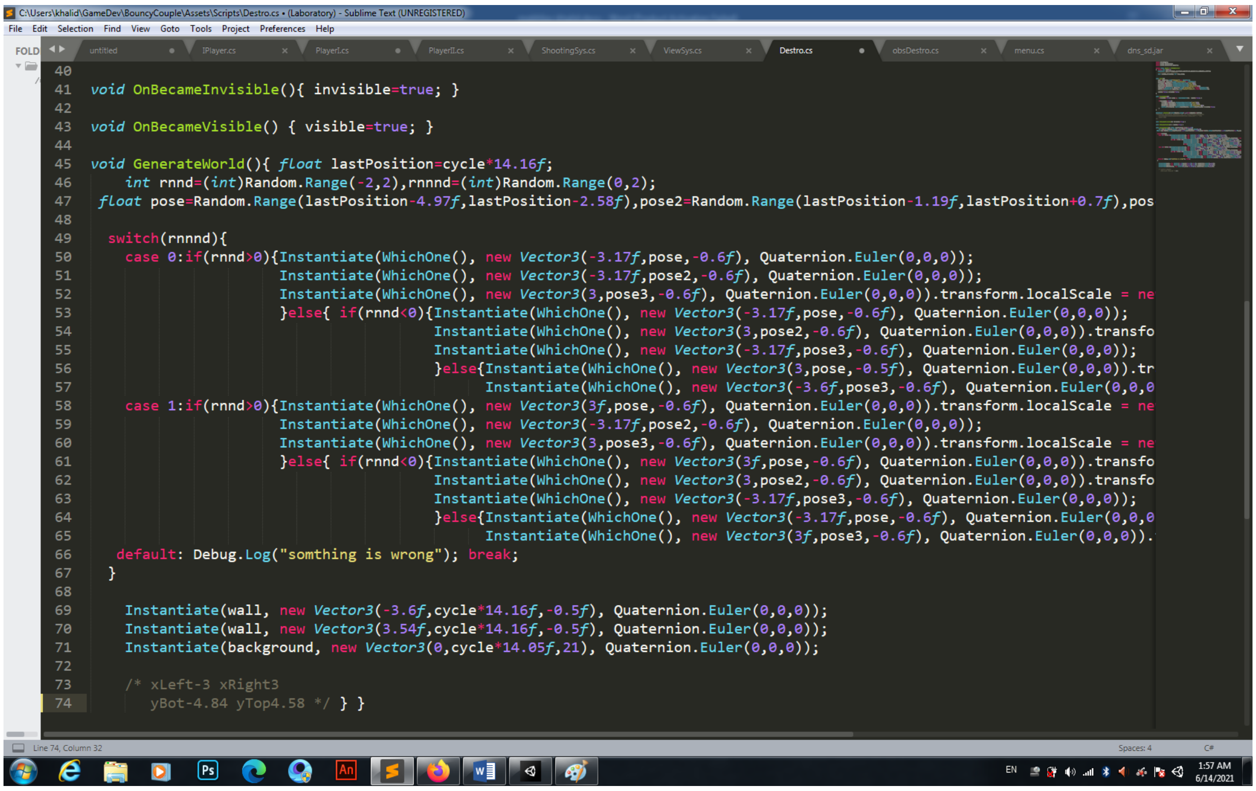


```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class Destro : MonoBehaviour
{  static int cycle=0;
   GameObject wall,firstWall,secondWall,backGround,background,obsBubble,obsFire;
   //List<GameObject> obstacles;
   bool visible,invisible; int rnnd,rnnnd;


void Start(){
   wall= Resources.Load("thisWall") as GameObject;
   background= Resources.Load("background") as GameObject;
   firstWall=GameObject.FindGameObjectsWithTag("wall")[0];
   secondWall=GameObject.FindGameObjectsWithTag("wall")[1];
   backGround=GameObject.FindGameObjectsWithTag("backGround")[0];
   obsBubble=(GameObject) Resources.Load("globalBgun_0", typeof(GameObject));
   obsFire=(GameObject) Resources.Load("globalfireGun_0", typeof(GameObject));
   //obstacles=new List<GameObject>();
   visible=false; invisible=false;
}

void FixedUpdate(){
    if(visible==true){ cycle++;  GenerateWorld();  visible=false; }

    if(invisible==true){
        firstWall=GameObject.FindGameObjectsWithTag("wall")[0];
        secondWall=GameObject.FindGameObjectsWithTag("wall")[1];
        backGround=GameObject.FindGameObjectsWithTag("backGround")[0];
        Destroy(firstWall); Destroy(secondWall); Destroy(backGround); invisible=false;
    }
}

GameObject WhichOne()=>(int)Random.Range(-1,1)<0?obsBubble:obsFire;
/*GameObject WhichOne(){ GameObject obj; int n=(int)Random.Range(0,2);
```

Figure 3.19 - Destro.cs part#1

By this way we manipulate those two variables that we use in the FixedUpdate in the conditions to know if we need to generate or destroy the walls and the back ground that create the world.

The generation process is described at the **GenerateWorld** method as shown at the figure 3.20.



```
void OnBecameInvisible(){ invisible=true; }

void OnBecameVisible() { visible=true; }

void GenerateWorld(){ float lastPosition=cycle*14.16f;
    int rnnd=(int)Random.Range(-2,2),rnnnd=(int)Random.Range(0,2);
 float pose=Random.Range(lastPosition-4.97f,lastPosition-2.58f),pose2=Random.Range(lastPosition-1.19f,lastPosition+0.7f),pos

  switch(rnnnd){
    case 0:if(rnnd>0){Instantiate(WhichOne(), new Vector3(-3.17f,pose,-0.6f), Quaternion.Euler(0,0,0));
                      Instantiate(WhichOne(), new Vector3(-3.17f,pose2,-0.6f), Quaternion.Euler(0,0,0));
                      Instantiate(WhichOne(), new Vector3(3,pose3,-0.6f), Quaternion.Euler(0,0,0)).transform.localScale = ne
                      }else{ if(rnnd<0){Instantiate(WhichOne(), new Vector3(-3.17f,pose,-0.6f), Quaternion.Euler(0,0,0));
                                        Instantiate(WhichOne(), new Vector3(3,pose2,-0.6f), Quaternion.Euler(0,0,0)).transfo
                                        Instantiate(WhichOne(), new Vector3(-3.17f,pose3,-0.6f), Quaternion.Euler(0,0,0));
                                        }else{Instantiate(WhichOne(), new Vector3(3,pose,-0.5f), Quaternion.Euler(0,0,0)).tr
                                              Instantiate(WhichOne(), new Vector3(-3.6f,pose3,-0.6f), Quaternion.Euler(0,0,0
    case 1:if(rnnd>0){Instantiate(WhichOne(), new Vector3(3f,pose,-0.6f), Quaternion.Euler(0,0,0)).transform.localScale = ne
                      Instantiate(WhichOne(), new Vector3(-3.17f,pose2,-0.6f), Quaternion.Euler(0,0,0));
                      Instantiate(WhichOne(), new Vector3(3,pose3,-0.6f), Quaternion.Euler(0,0,0)).transform.localScale = ne
                      }else{ if(rnnd<0){Instantiate(WhichOne(), new Vector3(3f,pose,-0.6f), Quaternion.Euler(0,0,0)).transfo
                                        Instantiate(WhichOne(), new Vector3(3,pose2,-0.6f), Quaternion.Euler(0,0,0)).transfo
                                        Instantiate(WhichOne(), new Vector3(-3.17f,pose3,-0.6f), Quaternion.Euler(0,0,0));
                                        }else{Instantiate(WhichOne(), new Vector3(-3.17f,pose,-0.6f), Quaternion.Euler(0,0,0
                                              Instantiate(WhichOne(), new Vector3(3f,pose3,-0.6f), Quaternion.Euler(0,0,0)).
   default: Debug.Log("somthing is wrong"); break;
  }

    Instantiate(wall, new Vector3(-3.6f,cycle*14.16f,-0.5f), Quaternion.Euler(0,0,0));
    Instantiate(wall, new Vector3(3.54f,cycle*14.16f,-0.5f), Quaternion.Euler(0,0,0));
    Instantiate(background, new Vector3(0,cycle*14.05f,21), Quaternion.Euler(0,0,0));

    /* xLeft-3 xRight3
       yBot-4.84 yTop4.58 */ } }
```

Figure 3.20 - Destro.cs part#2

Also at the figure 3.20 we can see the process of obstacles generation

```
void GenerateWorld(){ float lastPosition=cycle*14.16f;
    int rnnd=(int)Random.Range(-2,2),rnnnd=(int)Random.Range(0,2);
 float pose=Random.Range(lastPosition-4.97f,lastPosition-
2.58f),pose2=Random.Range(lastPosition-
1.19f,lastPosition+0.7f),pose3=Random.Range(lastPosition+2.12f,lastPositi
on+4.63f);

  switch(rnnnd){
    case 0:if(rnnd>0){Instantiate(WhichOne(), new Vector3(-
3.17f,pose,-0.6f), Quaternion.Euler(0,0,0));
                  Instantiate(WhichOne(), new Vector3(-
3.17f,pose2,-0.6f), Quaternion.Euler(0,0,0));
                  Instantiate(WhichOne(), new Vector3(3,pose3,-
0.6f), Quaternion.Euler(0,0,0)).transform.localScale = new Vector3(-
0.5f,0.5f, 1);
                  }else{ if(rnnd<0){Instantiate(WhichOne(), new
Vector3(-3.17f,pose,-0.6f), Quaternion.Euler(0,0,0));
```

```
                                             Instantiate(WhichOne(), new
Vector3(3,pose2,-0.6f), Quaternion.Euler(0,0,0)).transform.localScale =
new Vector3(-0.5f,0.5f, 1);
                                             Instantiate(WhichOne(), new
Vector3(-3.17f,pose3,-0.6f), Quaternion.Euler(0,0,0));

}else{Instantiate(WhichOne(), new Vector3(3,pose,-0.5f),
Quaternion.Euler(0,0,0)).transform.localScale = new Vector3(-0.5f,0.5f,
1);

Instantiate(WhichOne(), new Vector3(-3.6f,pose3,-0.6f),
Quaternion.Euler(0,0,0));}} break;
         case 1:if(rnnd>0){Instantiate(WhichOne(), new Vector3(3f,pose,-
0.6f), Quaternion.Euler(0,0,0)).transform.localScale = new Vector3(-
0.5f,0.5f, 1);
                           Instantiate(WhichOne(), new Vector3(-
3.17f,pose2,-0.6f), Quaternion.Euler(0,0,0));
                           Instantiate(WhichOne(), new Vector3(3,pose3,-
0.6f), Quaternion.Euler(0,0,0)).transform.localScale = new Vector3(-
0.5f,0.5f, 1);
                           }else{ if(rnnd<0){Instantiate(WhichOne(), new
Vector3(3f,pose,-0.6f), Quaternion.Euler(0,0,0)).transform.localScale =
new Vector3(-0.5f,0.5f, 1);
                                             Instantiate(WhichOne(), new
Vector3(3,pose2,-0.6f), Quaternion.Euler(0,0,0)).transform.localScale =
new Vector3(-0.5f,0.5f, 1);
                                             Instantiate(WhichOne(), new
Vector3(-3.17f,pose3,-0.6f), Quaternion.Euler(0,0,0));

}else{Instantiate(WhichOne(), new Vector3(-3.17f,pose,-0.6f),
Quaternion.Euler(0,0,0));

Instantiate(WhichOne(), new Vector3(3f,pose3,-0.6f),
Quaternion.Euler(0,0,0)).transform.localScale = new Vector3(-0.5f,0.5f,
1);}} break;
        default: Debug.Log("somthing is wrong"); break;
       }
```

```
         Instantiate(wall, new Vector3(-3.6f,cycle*14.16f,-0.5f),
Quaternion.Euler(0,0,0));
         Instantiate(wall, new Vector3(3.54f,cycle*14.16f,-0.5f),
Quaternion.Euler(0,0,0));
         Instantiate(background, new Vector3(0,cycle*14.05f,21),
Quaternion.Euler(0,0,0));

      }
```

At this stage I made I random system that create between two and three obstacles every cycle, and each generation we call the **WhichOne()** method as described below

```
      GameObject WhichOne()=>(int)Random.Range(-1,1)<0?obsBubble:obsFire;
```

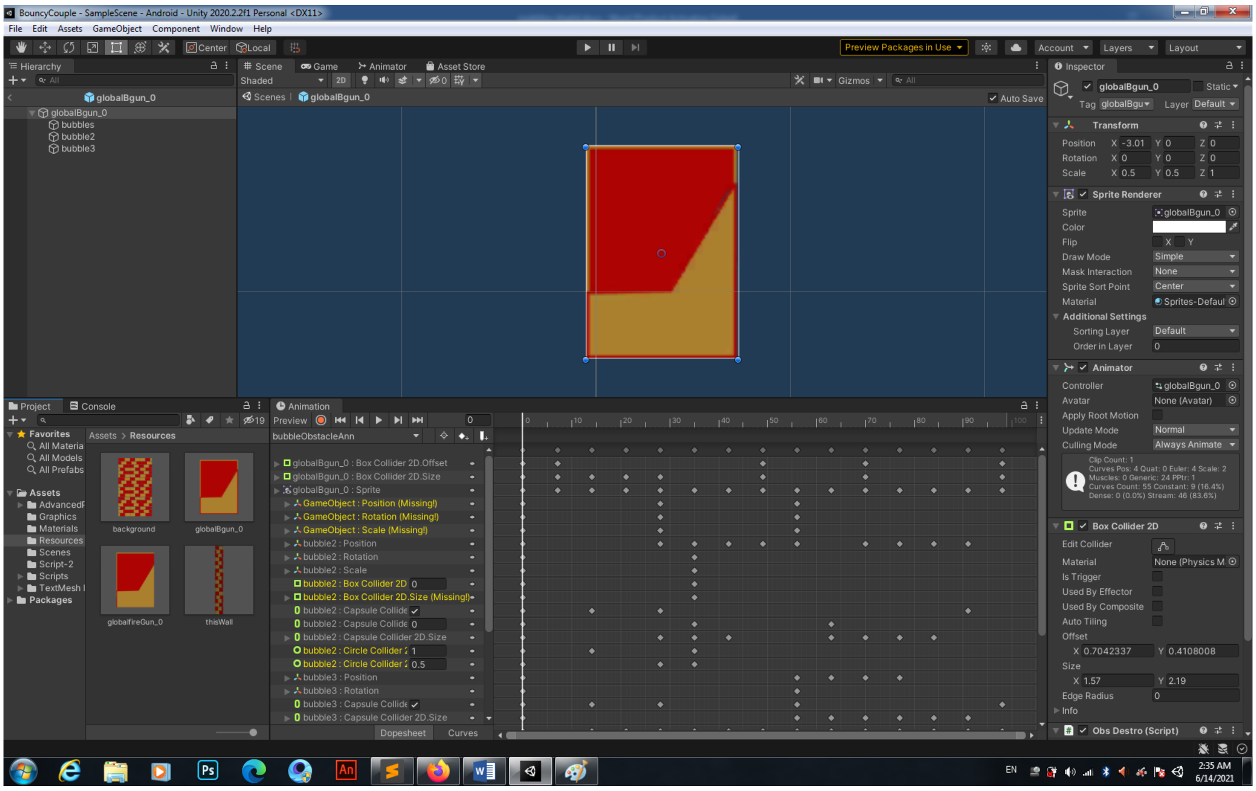
This line of code is the method that return one of the types of obstacles as a gameObject, and the rest of the generation is a fixed instantiation of tow walls and a background at aa fixed x axis and different y axis that's why the cycle variable should be static so every time the class is called for generation the cycle variable increment by +1.

And about the prefabs in unity system are a special type of component that allows fully configured GameObjects to be saved in the Project for reuse. These assets can then be shared between scenes, or even other projects without having to be configured again. This is quite useful for objects that will be used many times, such as platforms. A major benefit to Prefabs is that they are essentially linked copies of the asset that exist in the Project window. This means that changes made or applied to the original Prefab will be propagated to all other instances. This makes fixing object errors, swapping out art or making other stylistic changes very efficient.

Prefabs are created automatically when an object is dragged from the Hierarchy into the Project window.

Prefabs look quite similar to other objects that appear in the Project window. However, when selected, their file type will be a prefab. When the Prefab is selected, the Inspector displays all of the components that were configured on the original object.

As shown in the figure 3.21 a prefab globalBgun is selected.



BouncyCouple - SampleScene - Android - Unity 2020.2.2f1 Personal <DX11>

```
Inspector
globalBgun_0                      Static
Tag globalBgun      Layer Default

Transform
Position   X -3.01   Y 0     Z 0
Rotation   X 0       Y 0     Z 0
Scale      X 0.5     Y 0.5   Z 1

Sprite Renderer
Sprite              globalBgun_0
Flip                X  Y
Draw Mode           Simple
Mask Interaction    None
Sprite Sort Point   Center
Material            Sprites-Default
Additional Settings
  Sorting Layer     Default
  Order in Layer    0

Animator
Controller          globalBgun_0
Avatar              None (Avatar)
Apply Root Motion
Update Mode         Normal
Culling Mode        Always Animate
Clip Count: 1
Curves Pos: 4 Quat: 0 Euler: 4 Scale: 2
Muscles: 0 Generic: 24 PPtr: 1
Curves Count: 55 Constant: 9 (16.4%)
Dense: 0 (0.0%) Stream: 46 (83.6%)

Box Collider 2D
Edit Collider
Material            None (Physics Material 2D)
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset   X 0.7042337   Y 0.4108008
Size     X 1.57        Y 2.19
Edge Radius         0
Info

Obs Destro (Script)
```

```
Animation
bubbleObstacleAnn
globalBgun_0 : Box Collider 2D.Offset
globalBgun_0 : Box Collider 2D.Size
globalBgun_0 : Sprite
  GameObject : Position (Missing!)
  GameObject : Rotation (Missing!)
  GameObject : Scale (Missing!)
  bubble2 : Position
  bubble2 : Rotation
  bubble2 : Scale
  bubble2 : Box Collider 2D      0
  bubble2 : Box Collider 2D.Size (Missing!)
  bubble2 : Capsule Collider     ✓
  bubble2 : Capsule Collider     0
  bubble2 : Capsule Collider 2D.Size
  bubble2 : Circle Collider      1
  bubble2 : Circle Collider      0.5
  bubble3 : Position
  bubble3 : Rotation
  bubble3 : Capsule Collider     ✓
  bubble3 : Capsule Collider 2D.Size
```

Figure 3.21 - Unity obstacle prefab

3.4 Animation process

Also as we can see the prefab have already a lot of component predefined, and not just that also animation that ply directly after the prefab is instantiated.

At the animation window we can understand the animation unity system where we can add on the ordered sprites, defining the other stats of the component of the prefab.

And about the animation unity system we have another window named **animator**, at this window we can set the different animation stats (idle, spinning, grabbing and death) of the character. And define also the relations between them.

As we can see at the figure 15 we can understand that every animation stat we can create some variables which we can manipulate the animations also every animation scheme have two common states the first is **Entry** and the other is **Any State**

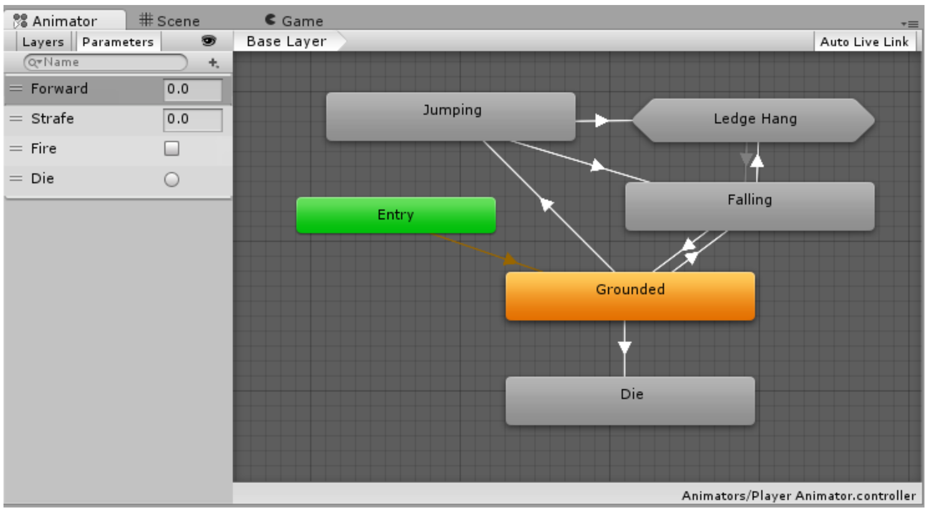


Figure 3.22 - Animator window

The entry state is the first state of any animation that start after the instantiation of the gameObject attached with this animation, also the Any State is a solution for those states that can start from a common possibility, like when we want to play the death animation we can play it while jumping or maybe running, from any state but just if the boolean variable **isDead** attached to this state is true.

I hope I succeeded to explain all the process of a simple 2D game development and make the big steps clear and simple, the creation of games is to fun but also too hard and need a real team with different skills.

4 LIFE SAFETY, FUNDAMENTALS OF LABOR PROTECTION

4.1 The electromagnetic fields effects

Effects of emf on cancer.

Previously, epidemiological studies have been done whether children with chronic exposure to ELF-EMF or RF-EMF develop childhood leukemia and in adults, brain tumors and leukemia may occur [5, 6]. Although there were some uncertainties, both epidemiological studies found that there is unlikely to be a material increase in risk of adult brain tumors or childhood leukemia resulting from the exposure to either ELF-EMF or RF-EMF. In addition, other studies have not found a direct evidence for the increase in the incidence of childhood leukemia due to ELF-EMF exposure [7, 8]. Moreover, there is no direct correlation between acute lymphoblastic leukemia in children and exposure to ELF-EMF in the home [7]. Therefore, a standard health risk assessment process, the WHO Task Group of scientific experts concluded that there are no substantive health issues related to ELF-EMFs at levels generally encountered by members of the public.

However, it has been hypothesized that a variety of neurological influences may arise as a result of EMF exposure due to the location of the cellular phone and the proximity of the cranial nervous system. Therefore, a lot of concern is focused on possible carcinogenesis of the cranial nervous system by exposure to RF-EMF [9]. An epidemiological study claimed that using cell phones for one hour a day for more than 10 years could increase the risk of tumors [10]. In addition, cell phone users have an increased risk of malignant gliomas, particularly those with acoustic neuromas. Moreover, many studies have reported associations between RF-EMF and brain tumors. In contrast, there are some studies claiming no association between brain cancer and cell phone usage or due to nearby cellular base stations. Another study showed that there was also no association between cancer and infants’ risk of exposure to cellular base stations during pregnancy. From this point of view, the observed results to date suggest that the association between the possible carcinogenicity of EMF and the cranial nervous system is complicated by a wide range of confounding variables. There is no clear evidence to support the causal relationship between increased carcinogenicity following exposure to EMFs. Despite these controversies, the World Health Organization has classified RF-EMFs as ‘possibly carcinogenic to humans’. However, the classification of RF-EMFs as possible carcinogens has yet to come to a clear conclusion among scientists. This is due to the fact that only 30 years have passed since the mobile phone has been used in earnest, it needs decades of exposure, and further epidemiological analysis to come to any conclusions.

Effects of emf on learning and memory.

It has been hypothesized that various neurological effects may arise as a result of RF-EMF exposure due to the proximity of the cranial nervous system during cellular phone use. These neurological effects include headache, changes in sleep habits, changes in electroencephalogram, and changes in blood pressure but there are many inconsistent results. Neurological cognitive disorders, such as headache, tremor, dizziness, loss of memory, loss of concentration and sleep disturbance due to RF-EMF have also been reported by several epidemiological studies. The exposure levels of RF-EMF, commonly encountered in public environments have been found to be nondetrimental level to human, however with respect to the amount of exposure to RF-EMFs on the cranial nervous systems, a significant amount of research has focused on rodent behavioral disorders, particularly learning and memory deficits, after RF-EMF exposure under various conditions.

The radial maze and Morris water maze test showed that learning and memory functions were reduced in rats exposed to 2,450 MHz EMF [11], but no changes of working memory were observed in the radial maze test following whole body exposure for 45 minutes for 10 days at 2,450 MHz, 0.6 W/kg SAR and in the Y-maze, Morris water maze, and novel object recognition memory test after exposed to 1950 MHz electromagnetic fields (SAR 5 W/kg, 2 h/day, 5 days/week) for 3 months. Recognition memory was studied using the object recognition test using a head-only exposed mouse (900 MHz GSM, 1 and 3.5 W/kg SAR). In this study, there was no effect on learning and memory at low SAR levels, but at high SAR levels, only some of the exploration activities were changed. Although exposure to RF-EMFs could affect cognitive function such as spatial learning and memory loss in both humans and in animals, direct evidence for the effects of RF-EMFs on these functions remains unclear. The hippocampus is involved in spatial memory and learning processes and low-intensity RF-EMFs at 700 MHz can alter electrical activity in hippocampal slices of the rat brain. Similarly, exposure to 1,800 MHz (15 min per day for 8 days, SAR 2.40 W/kg) has been reported to reduce excitatory synaptic activity of cultured hippocampal neurons. Although the water maze test results showed increased behavioral performance, there were no changes in spatial memory performances shown by both the open field or the plus maze tests, as well as in acoustic startle experiments in juvenile rats exposed to a 900 MHz for 5 weeks (2 hours per day, 5 days per week, SAR 3 W/kg).

Recently, with regard to the effect of RF-EMFs on cognitive function, it has been found that exposure induced the improvement in cognitive behavior of triple transgenic mice (3xTg-AD), that have a cognitive impairment such as in human Alzheimer’s disease. This experiment was performed with a Wi-Fi type, 2.40 GHz RF signal for 2 hours a day for 1 month at 1.60 W/kg SAR. Experimental results suggest that exposure of RF-EMF can lead to effective memory recovery in cognitive impairment in an experimental animal with loss of cognitive function caused by Alzheimer’s disease. Despite numerous studies, it remains unclear if RF-EMF exposure is a risk for cognitive function, including memory. However, transgenic Alzheimer’s mice with long-term RF-EMF exposure for more than 8 months have been reported to improve cognitive abilities. These series of experimental results suggest that exposure to RF-EMFs can improve memory in Alzheimer’s disease, which is based on reduced response time, less anxiety, but no effect on exercise activity, body weight or body temperature.

4.2 First-aid care provision for a broken bone

If you suspect that someone has a broken bone, provide first-aid treatment and help them get professional care: Stop any bleeding: If they’re bleeding, elevate and apply pressure to the wound using a sterile bandage, a clean cloth, or a clean piece of clothing.

Immobilize the injured area: If you suspect they’ve broken a bone in their neck or back, help them stay as still as possible. If you suspect they’ve broken a bone in one of their limbs, immobilize the area using a splint or sling.

Apply cold to the area: Wrap an ice pack or bag of ice cubes in a piece of cloth and apply it to the injured area for up to 10 minutes at a time.

Treat them for shock: Help them get into a comfortable position, encourage them to rest, and reassure them. Cover them with a blanket or clothing to keep them warm.

Get professional help: Call ambulance or help them get to the emergency department for professional care.

If the person doesn’t appear to be breathing, is unconscious, or both, call ambulance for medical help and begin CPR. You should also call ambulance if:

– you suspect they’ve broken a bone in their head, neck, or back;
– the fractured bone has pushed through their skin;
– they’re bleeding heavily.

Otherwise, help them get to the emergency department by car or other means so a doctor can diagnose their condition and recommend appropriate treatment.

CONCLUSION

At the end I get closer and closer to my life love which is game development, after I master the artificial intelligence process I will be able to create more powerful games with more creative ideas, the game industry is a huge opportunity and can get much huger with the development of technologies.

The plan of my objective is taking control of the mobile game industry first which it can be happened with low budget and less work, but still need a lot of creativity.

After this I’m planning to get a famous studio by a masterpiece game which is going to allow me to enter the higher industry like pc games or why not get a contract with famous console.

And I can’t forget to thanks all the professors who helped me to reach the knowledge I need to make my dreams true, like PhD. M. Fryz and our dean Baran also the university’s iron man mr.Zoloti and my supervisor Mr.Bodnarchyk without forgetting the help and attention of the dean office team and I’m mentioning one person who really can do the work of a team our dear Mrs. Oksana

With all my respect to all the rest of the great minds around this university, Mokhliss Khalid.

REFERENCES

1. https://www.brainbalancecenters.com (Jan, 2020)
2. https://www.healthgrades.com (Feb, 2020)
3. https://www.healthline.com (Jun, 2020)
4. https://docs.unity3d.com/ScriptReference/index.html (Jan, 2020)
5. Linseisen, Jakob, et al. "Meat consumption in the European Prospective Investigation into Cancer and Nutrition (EPIC) cohorts: results from 24-hour dietary recalls." *Public Health Nutrition* 5.6b (2002): 1243-1258.
6. Stephens, Philip J., et al. "Massive genomic rearrangement acquired in a single catastrophic event during cancer development." *cell* 144.1 (2011): 27-40.
7. Wong, F. Lennie, et al. "Cancer incidence after retinoblastoma: radiation dose and sarcoma risk." Jama 278.15 (1997): 1262-1267.
8. Rana, Subinoy, et al. "Array-based sensing of metastatic cells and tissues using nanoparticle–fluorescent protein conjugates." *ACS nano* 6.9 (2012): 8233-8240.
9. Hardell, Lennart, Michael Carlberg, and Kjell Hansson Mild. "Case-control study on cellular and cordless telephones and the risk for acoustic neuroma or meningioma in patients diagnosed 2000–2003." Neuroepidemiology 25.3 (2005): 120-128.
10. Hardell, Lennart, et al. "Long-term use of cellular phones and brain tumours: increased risk associated with use for⩾ 10 years." Occupational and environmental medicine 64.9 (2007): 626-632.
11. Bochar, Daniel A., et al. "BRCA1 is associated with a human SWI/SNF-related complex: linking chromatin remodeling to breast cancer." Cell 102.2 (2000): 257-265.